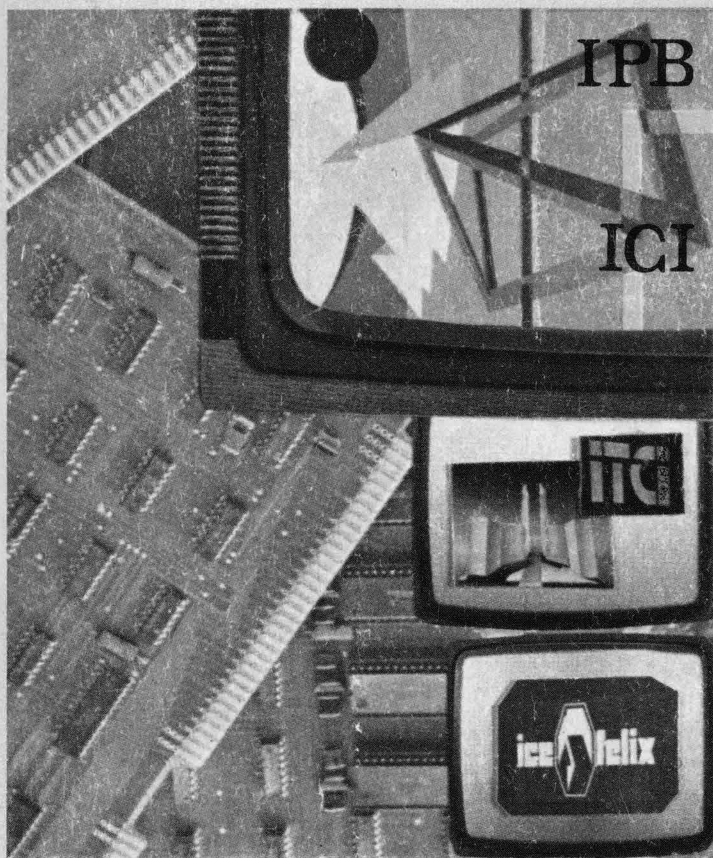
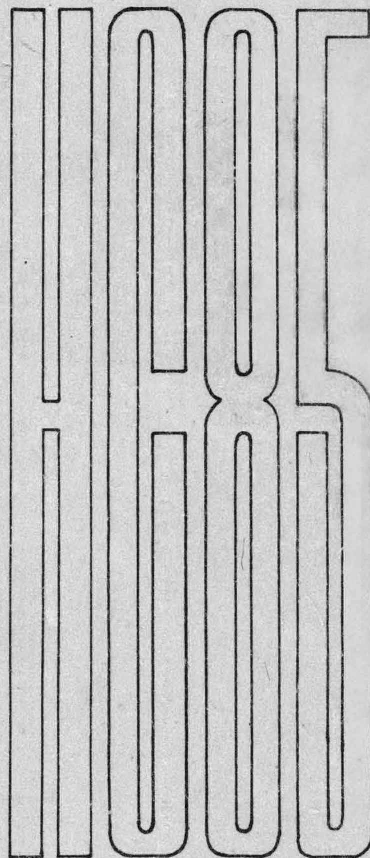
A. Petrescu
N. Țăpuș T. Moisa Gh. Rizescu
V. Hărăbor M. Mârșanu Tr. Mihu

abc de CALCULATOARE PERSONALE și... nu doar atît...

1





EDITURA TEHNICĂ

AUTOMATICĂ
INFORMATICĂ
ELECTRONICĂ
MANAGEMENT

SERIA INIȚIERE

BIBLIOTECA DE

Automatică — Informatică — Electronică — Management

SERIA INIŢIERE

E. VASILIU
INIŢIERE ÎN DISPOZITIVELE SEMICONDUCTOARE
D. STANOMIR
INIŢIERE ÎN ELECTROACUSTICA
W. TRUSZ
ABC-UL REPARĂRII RADIORECEPTOARELOR
Traducere din lb. polonă (Ciclul ABC-uri)
A. POPA
ABC DE PROTECŢIA MUNCII (ciclul ABC-uri)
MARGARETA DRĂGHICI
INIŢIERE ÎN COBOL
STELIAN NICULESCU
INIŢIERE ÎN FORTRAN
PAUL CONSTANTINESCU ŞI ZAHARIA NICOLAE
INIŢIERE ÎN ORGANIZAREA ŞI PROIECTAREA SISTEMELOR DE CONDUCERE
I. CREŢU
INIŢIERE ÎN ESTETICA PRODUSELOR (ciclul ABC-uri)
E. AISBERG
ABC DE RADIO ŞI TELEVIZIUNE
Traducere din limba franceză
J. D. WARNIER, B. MI FLANGAN
INSTRUIRE ÎN PROGRAMARE
Traducere din limba franceză
I. H. BERNHARD, B. KNUPPERTZ
INIŢIERE ÎN TIRISTOARE
Traducere din limba germană
W. DEPPEERT, K. STOLL
INIŢIERE ÎN PNEUMOAUTOMATICA
Traducere din limba germană
INIŢIERE ÎN RADIOELECTRONICA CUANTICA
V. POPESCU
INIŢIERE ÎN CIBERNETICA SISTEMELOR INDUSTRIALE
I. PAPADACHE
AUTOMATIZĂRI INDUSTRIALE, INIŢIERE, APLICAŢII
ŞT. NICULESCU
FORTRAN, INIŢIERE ÎN PROGRAMARE STRUCTURATA
J. FORRESTER
PRINCIPIILE SISTEMELOR: TEORIE ŞI AUTOINSTRUIRE PROGRAMATĂ
Traducere din lb. engleză — S.U.A.
P. DRANSFIELD, D. F. HABER
INSTRUIRE PROGRAMATĂ ÎN METODA LOCULUI RĂDACINILOR
Traducere din lb. engleză — S.U.A.
D. RODDY
INIŢIERE ÎN MICROELECTRONICA
Traducere din lb. engleză
NICULESCU Cl., IOSIF M.
INIŢIERE ÎN COMUNICAŢIILE PRIN FIBRE OPTICE
CSABAI DÁNIEL
TEHNICA SONORIZĂRII (traducere din lb. maghiară)
MITROFAN GH., PFLANZER G.
INIŢIERE ÎN TELEVIZIUNEA ÎN CULORI
RADU NEGOESCU
INIŢIERE ÎN ELECTRONICA BIOMEDICALA
RADU NEGOESCU
INSTRUMENTAŢIA ELECTRONICA BIOMEDICALA
A. PETRESCU ş.a.
TOTUL DESPRE... CALCULATORUL PERSONAL aMIC
I. DUMITRAŞCU ş.a.
ÎNVĂŢĂM FORTRAN... CONVERSÎND CU CALCULATORUL
I. DUMITRAŞCU
ÎNVĂŢĂM COBOL... CONVERSÎND CU CALCULATORUL
I. DUMITRAŞCU
ÎNVĂŢĂM MICROELECTRONICA INTERACTIVA
M. PATRUBANY
TOTUL DESPRE... MICROPROCESORUL Z 80

Prof. dr. ing.
Adrian Petrescu

şef catedră dr. ing.
Nicolae Ţăpuş

şef lucrări dr. ing.
Trandafir Moisa

prof. emerit liceu mat.
Gheorghe Rizescu

matematician
Viorica Hărăbor

drd. ing.
Mihai Mârşanu

drd. ing.
Traian Mihu

abc de CALCULATOARE PERSONALE şi... nu doar atît...

Coordonare generală: Prof. dr. ing. ADRIAN PETRESCU

Studiu introductiv: Academician Prof. dr. NICOLAE TEODORESCU

Volumul 1

EDITURA TEHNICĂ
Bucureşti, 1990

CONTRIBUŢIA AUTORILOR

A. Petrescu: 3, 4, 5, 6, 7
N. Tăpuş: 11, 12 p.
T. Moisa: 10, 12 p
Gh. Rizescu: P-D-E p
V. Hărăbor: 8, 9.
M. Mârşanu: 1.1, 1.2, 2.1, 2.2, 2.3
D. Mihu 1.3, 1.4, 2.4, p

RECENZII: Dr. ing. **PAUL CONSTANTINESCU**, dr. ing. **LIVIU DUMITRAŞCU**
dr. ec. **GHEORGHE SABAU**, ing. **CONSTANTIN MÎNDRULEANU**

REDACTOR: ing. **PAUL ZAMFIRESCU**

Mulţumiri întregului colectiv al Întreprinderii poligrafice Sibiu (Director: ing. Petre Pascu; ing. şef: Gheorghe Tămaş; producţie: ing. Ioan Şiandru (şef producţie), Eugenia Oleksik; **maiştri în diverse compartimente:** Ioan Simtion (m. princ.), Ioan Stroie, Vasile Ienciu, Iancu Popescu, Ioan Rodean, Mircea Pascu, dar mai ales, celor care — cu deosebire — au contribuit direct la realizarea lucrării: (**Tastere monotyp:** Marioara Suciu, Emilia Lungu, Livia Mihu; **monoturnare:** Mihai Schuster, Ioan Gîndilă, Ilie Lupu, Ionel Bica; **linotip:** Florentina Murărescu, **paginare:** Dan Koss, ajutat de Mioara Maşca, Ana Berea, Ana Străjan: **pregătire offset:** Nagy Sibile, Fănica Popovici, Ileana Pădureanu, Marcela Hiesch, Daniela Boantă, Cornelia Spinei, Mariana Băilă, Victoria Barbulat, Mirela Popa; **tipar offset:** Nicolae Prişcă, Nicolae Vinersar, ajutaţi de Ioan Funariu, Octavian Moga; **corectură:** Marcela Tamaş, Alina Iliu, Maria Cheleţ, Maria Stoişor; **legătorie:** Jenica Matei, Elena Lăcătuş, Ana Szylaghi, Paraschiva Giuşcă, Ştefan Nanaşi, Cornel Banea Cu scuzele de rigoare pentru omisiunile nedorite, reparabile în al doilea tiraj.

ISBN 973-31-0013-7

ISBN 973-31-0014-5

COPERTE: Arh. **SILVIA PINTEA**
DESENE: Arh. **IULIAN MIHAESCU**

TEHNOREDACTARE: (iniţială) VICTORIA UNGUREANU;
adaptări şi completări 1990 — redacţia

Bun de tipar 27.04.1990; Coli de tipar 21,5.

Cărţile s-au realizat la Întreprinderea poligrafică Sibiu

Studiu introductiv

Un titlu atractiv, îmbietor, un colectiv de autori specialişti de concepţie şi proiectare, de profesori competenţi şi inimoşi este cartea de vizită cu care se prezintă modest şi ispititor „ABC de calculatoare personale ... şi nu doar atît“, *lucrare de mare atractivitate, de o amploare care corespunde în cea mai largă măsură adagiului „*... şi nu doar atît“, *fără a se diminua cu nimic ponderea primei părţi a titlului.*

Startul — dar şi finalul cărţii — sînt date într-un original PROLOG—DIALOG—EPILOG, care este o elaborare din 1990, menită de a reflecta probleme cardinale ale concepţiilor autorilor şi editorului, în lumina gîndirii revoluţionare ce va anima, în mod sigur, informatizarea amplă în învăţămîntul şi educaţia societăţii româneşti.

Concepută cu o structură bogată şi complexă, cuprinsă în două volume (cu trei casete magnetice asociate unei părţi a tirajului), lucrarea beneficiază de contribuţia unor autori prestigioşi: trei dintre aceştia: prof. Adrian Petrescu *şi şefii de lucrări* Nicolae Ţăpuş *şi* Trandafir Moisa, *din Catedra de Calculatoare a Institutului Politehnic Bucureşti au obţinut în 1987 premiul I la Concursul naţional de creaţie ştiinţifică şi tehnică pentru calculatorul personal-profesional Felix-PC, prof. Adrian Petrescu, colectiv IPB şi ICE obţinînd în plus premiul II pentru calculatorul personal HC 85. Adăugăm că prof. emerit Gh. Rizescu a obţinut, în 1987, în cadrul aceluiaşi concurs, premiul I pentru comunicarea privind folosirea calculatorului în învăţămîntul liceal. Şi ceilalţi autori au merite distincte în informatică şi tehnică de calcul:* Mihai Mârşanu — *de la Institutul de Tehnică de Calcul,* Viorica Hărăbor *şi* Nicolae Badea *de la Institutul de Cercetări pentru Informatică;* Tr. Mihu, *Eugen Dobrovie şi Viorel Cososchi — Întreprinderea de Calculatoare Electronice; C. Hărăbor este profesor la liceul D. Bolintineanu.*

Cartea beneficiază din punct de vedere editorial şi de competenţa ca autor a inginerului Paul Zamfirescu, din redacţia de informatică şi tehnică de calcul a Editurii Tehnice, el însuşi autor cunoscut în domeniul automaticii şi informaticii.

Lucrarea este împărţită în 11 părţi, repartizate 5 în primul volum şi 6 în cel de-al doilea. Aceste părţi se împart în 25 de capitole, subdivizate la rîndul lor, marea majoritate, în paragrafe care sînt fiecare consacrate unei teme importante atît prin ea însăşi cît şi prin integrarea sa în contextul general al întregii lucrări; un amplu PROLOG—DIALOG—EPILOG, o bibliografie bogată şi anexe de actualizare completează fericit ansamblul.

Partea I *descrie atrăgător şi substanţial „Evoluţia maxi/mini/micro calculatoarelor şi a calculatoarelor personale în ţara noastră şi pe plan mondial.“ Este o temă care cîştigă imediat pe orice cititor, nu numai pe elevii care ar voi să se iniţieze în tehnicile de calcul, cu vedeta actuală a calculatoarelor electronice, calculatorul personal, care trebuie să se răspîndească, în grabă, şi în învăţămîntul nostru primar, gimnazial, liceal şi superior.*

Lectura celor două capitole ale acestei părți ne întoarce în tunelul timpului, la anii apariției calculatoarelor electronice, la minunile pe care ni le evocau presa și imaginația noastră, la vertiginoasa lor evoluție care trimite în preistoria acestei tulburătoare și covîrșitoare invenții.

Contribuții și priorități românești în matematică, cibernetică, electronică, tehnică de calcul și informatică, evocate în primul capitol, ne amintesc de anii în care se frămînta pe plan mondial ideea realizată prin inventarea și punerea în aplicație a calculatorului electronic, prin conjugarea eforturilor în aceste domenii.

Azi avem o industrie proprie de calculatoare, iar ideea autorilor de a releva stadiul acesteia și al informaticii în țara noastră este binevenită, ca și cea de a prezenta succint unele aplicații prestigioase ale calculatoarelor în societatea modernă.

Capitolul 2 ne aduce pe planul actualității în domeniul invocat de însuși titlul lucrării, „Evoluția calculatoarelor personale și a microcalculatoarelor personale", începînd cu sublinierea a două elemente de referință. Primul *este că această carte este printre primele consacrată învățării utilizării calculatorului personal în țara noastră, și că este prima dedicată integral primului calculator românesc HC-85 compatibil cu un model internațional, Sinclair Spectrum* (Marea Britanie).

Al doilea *este faptul că,dintru-început, cititorul este inițiat în structura și funcționalitatea calculatorului personal, pus la curent cu rolul și importanța microelectronicii pe plan mondial, cu locul ocupat în prezent de calculatoarele personale și de microcalculatoare în societatea contemporană, precum și cu dezvoltarea actuală a industriei românești în aceste sectoare.*

Partea a II-a *se ocupă de „Calculatoare numerice. Realizare fizică. Baze aritmetice și logice" și este o introducere matematică, împlinită prin realizările fizice ale funcțiilor și circuitelor logice combinaționale și secvențiale. Aici găsim un capitol privitor la bazele aritmetice și altul la logica matematică și circuitele logice ale calculatorului numeric.*

O dată cu **partea a III-a** *este abordat studiul calculatorului personal HC85, din puctul de vedere al structurii și componentelor, al operării și programării. Sînt comparate structura și modul de operare ale calculatorului numeric cu cele ale lui HC 85, dîndu-se și elemente de programare în limbajul algoritmic.*

Pe măsură ce se înaintează în contextul primului volum se accentuează obiectivele studiului calculatorului personal HC 85. Astfel, în **partea a IV-a** *este abordată problema programării în limbajul BASIC, legat intim de acest calculator, dîndu-se caracteristicile și elementele acestui limbaj. Prezentarea este sistematică și gradată, cu grija deliberată de a fi accesibilă unor începători fără experiență, dar cu acea curiozitate care caracterizează pe copii și adolescenți. Găsim aici, după noțiunile introductive, paragrafe privitoare la tastatură, mod de lucru și alfabet.*

Capitolul 9 este destinat prezentării detaliate a instrucțiunilor limbajului BASIC. Cititorului i se atrage atenția că folosirea calculatorului este posibilă numai prin intermediul instrucțiunilor pentru care a fost proiectat, exemplificîndu-i-se semnificația noțiunii de instrucțiune, care cuprinde cuvîntul cheie și unul sau mai multe argumente. Prin aceasta se abordează cerința esențială a acordării treptate și sistematice a modului de gîndire umană cu cel al calculatorului.

Se ajunge astfel la **partea a V-a** *unde cititorul este inițiat în „Programarea în limbajul LOGO pe calculatorul HC 85". Capitolul 10 își asumă sarcina de a dezvălui caracteristicile și elementele limbajului LOGO, debutînd printr-o scurtă introducere în care se discută locul și importanța cîștigate de calculatoare în viața societății de azi, subliniindu-se cele ale calculatorului personal. Accesibilitatea deose-*

bită a acestuia este comparată, foarte sugestiv, cu apariția tiparului.

Evoluția limbajelor de programare, al căror număr a trecut de o sută, a condus și la limbajele BASIC și LOGO. Primul permite ca programatorul să „vorbească", să converseze cu calculatorul, iar cel de-al doilea se prezintă ca un limbaj conceput pentru învățare, dar el are și o semnificație mai profundă, aceea de a fi un limbaj de învățare a unui mod de gîndire. El își are originea în cercetările privitoare la știința calculatoarelor, care a promovat gîndirea secvențială, algoritmică, în special în cele ce adîncesc gîndirea artificială, iar din punct de vedere uman în cercetările lui Piaget cu privire la dezvoltarea gîndirii la copii. Autorii răspund la întrebarea „Ce este *LOGO?", punînd în evidență principalele caracteristici ale acestui limbaj, ca limbaj de programare. Apoi se consacră un paragraf special locului lui LOGO în școli, în care se atrage atenția asupra inițierii începătorilor și sarcinilor profesorilor de LOGO, arătîndu-se că programarea este pentru mulți oameni o modalitate importantă de a învăța să înveți, un mod de explorare intelectuală. Paragraful, care este o frumoasă pledoarie pentru LOGO, se încheie cu observația că microcalculatoarele produse în țara noastră, HC 85 și FELIX PC, ca și cele personale compatibile cu APPLE, oferă posibilitatea de a se folosi limbajul LOGO.*

Urmează folosirea lui LOGO pe HC-85, cu aspecte privitoare la utilizarea tastaturii, apoi reguli gramaticale ale acestui limbaj și obiectele sale: numere, cuvinte, liste, delimitatori, variabile. În continuare se dau reguli privind lucrul cu proceduri, proceduri cu intrări, tipuri de proceduri, introducerea și editarea acestora. Găsim, de asemenea, un paragraf pentru expresii condiționate, altul pentru linii complexe și în final un glosar de termeni LOGO. Remarcăm și întinderea deosebită a acestui capitol, foarte instructiv și convingător.

Procedurile implicite, denumite primitive, sînt la baza definirii procedurilor de descriere a algoritmilor diverselor programe. Autorii consacră un amplu capitol, 11, primitivelor limbajului LOGO, dînd exemplificări variate și binevenite.

Se începe cu caracteristicile generale ale primitivelor, urmate de primitivele LOGO pentru controlul penelului și ecranului în regim grafic. Apoi sînt prezentate, rînd pe rînd, primitive pentru schimbarea stării penelului, cu specificarea că în limbajul LOGO există două tipuri de primitive: comenzi și operații.

Cu aceste accepțiuni, urmează primitive care specifică starea penelului, apoi cele pentru utilizarea peniței electronice și a ecranului, cele pentru specificarea stării peniței și a ecranului, cele pentru controlul ecranului în regim alfa-numeric. Se trece apoi la primitive LOGO, care specifică operații matematice și logice. unde se disting operatori aritmetici și logici, primitive pentru operații aritmetice și pentru operații logice.

Alte tipuri de primitive LOGO sînt cele pentru lucrul cu cuvinte și liste, pentru preluarea unor elemente din cuvinte sau liste, pentru concatenarea și pentru examinarea acestora. Apoi, găsim primitive pentru comunicația cu echipamentele de intrare/ieșire, pentru citirea informației din exterior și altele pentru ecran.

Alte categorii sînt cele pentru generarea de sunete. Urmează primitive de asigurare a ramificației în program, cum sînt cele pentru ramificarea condiționată, pentru întreruperea procedurilor, pentru execuția și repetarea unei liste de instrucțiuni, pentru lucrul cu fișiere, pentru încărcarea de pe casetă, pentru salvarea pe casetă, pentru controlul imprimantei și tipărirea informației de pe ecran, pentru lucrul cu microdriverul, pentru examinarea spațiului de lucru și pentru specificarea obiectelor și primitivelor LOGO. Din această prezentare de primitive și din specificarea amănunțită a obiectivelor lor ca operații și comenzi, se desprinde destul de pregnant caracterul de limbaj de învățare, pentru cine vrea să învețe cum se poate

învăța cu calculatorul, prin transpunerea conversațională a gîndirii umane în cea a calculatorului, prin folosirea limbajului LOGO.

Desigur că mințile cele mai susceptibile de a folosi această modalitate de dezvoltare a propriei lor gîndiri sînt cele ale copiilor la vîrstele cînd școala își ia sarcina de a le deștepta și întemeia gîndirea logic-deductivă, la care se ajunge pe la vîrsta de 10—11 ani, desăvîrșirea începînd pe la cea de 15—16 ani și continuîndu-se cu eforturi susținute toată viața.

În capitolul 12, „Tehnici de programare în LOGO“ vom găsi procedurile prezentate în capitolul 10, cu accentul pus pe algoritmi, cu precizarea că elaborarea unui program implică atît dezvoltarea algoritmului cît și implementarea acestuia într-un limbaj de programare, operațiile fiind structurate pe două nivele, de care se vorbește permanent în informatică: nivelul logic și nivelul fizic. Vom găsi în capitol proiectarea programelor, ilustrată prin exemple, fiind apoi subliniată proprietatea de recursivitate în limbajul LOGO, care este o modalitate de a scrie proceduri care se apelează pe ele însele.

Capitolul este completat cu numeroase aplicații atractive ca exemple: grafice sumarea a două numere aleatoare, ordonarea alfabetică, conversia numerelor naturale dintr-o bază în alta, sau desenarea interactivă. Acestea sînt, fără excepție, menite totdeauna să cîștige pe începători, care își verifică în mod fericit puterile și se simt atrași spre însușirea cît mai temeinică a tehnicilor, chiar dificile, de programare.

O mențiune specială în încheierea acestui prim volum trebuie făcută cu privire la faptul că lucrarea prezintă pentru prima dată în literatura noastră de specialitate limbajul LOGO, ceea ce aduce un serviciu deosebit de prețios celor ce vor avea sarcina sau plăcerea de a preda folosirea calculatoarelor personale în procesul instructiv-educativ.

Arhitectura lucrării consacră cel de-al doilea volum cu precădere folosirii calculatorului personal HC 85 în învățămînt și educație.

Ministerul de specialitate (în prezent Ministerul Învățămîntului), *pornind de la necesitățile obiective de a se introduce informatica în perfecționarea procesului de învățămînt, a aprobat* — foarte timid — *introducerea în cadrul programei de matematică a unor capitole de informatică și programarea calculatoarelor la clasele IX—XII, cu începere din anul școlar 1987—1988. Și în cărțile de clasa a VII-a a apărut, în 1989, informatica.*

Editura Tehnică, publicînd în 1985, în cadrul aceleiași redacții (după cărțile din 1984 privind microcalculatoarele M118) cărțile de mare tiraj „Totul despre... calculatorul personal aMIC“, s-a corelat cu apariția primelor calculatoare personale românești (aMIC și Prae); continuînd în 1989 cu „Totul despre... microprocesorul Z 80, 2 vol.+casetă pe aMIC și PRAE), cu „Totul despre... BASIC, 2 vol.“, în tiraje de zeci de mii de exemplare a recunoscut implicit amploarea instruirii în informatică și tehnică de calcul, la toate nivelele.

În condițiile Revoluției din decembrie '89 *necesitatea lucrării a devenit stringentă. De aceea, vedem în realizarea ei un răspuns menit să se înscrie printre acțiunile care marchează o cotitură în pregătirea și educația elevilor noștri.*

Volumul al II-lea al lucrării are acest scop, dar ca și primul spune ceva mai mult, fiind deosebit de prețios nu numai elevilor, dar și profesorilor lor, și chiar oricui își dă seama că informatica cere și oferă un nou mod de gîndire, tot atît de indispensabil ca și abecedarul, cartea de citire și aritmetica primilor pași în viață ai copiilor întregii omeniri. Este un volum la fel de dens ca primul.

Partea a VI-a *tratează „Microcalculatorul HC 85 în procese industriale.*

Pachete de programe de aplicații pentru HC 85". Vom găsi aici un capitol privitor la proiectarea interfețelor pentru echipamente nestandard, cu aplicații în măsurători și conducerea microroboților industriali. Următorul capitol oferă pachete de programe aplicative: Grafica 3—D, Baze de date, Tabelare electronică, Prelucrarea textelor, realizate în străinătate și larg aplicate și în țara noastră.

Partea a VII-a *abordează problemele puse de calculatorul personal în procesul de învățămînt, ca locul informaticii în învățămîntul liceal și asistarea procesului de învățămînt cu calculatorul, ridicînd și pe cea a tipurilor de laboratoare pentru procesul de învățămînt, unde se prezintă în detaliu organizarea spațiului școlar respectiv. Ceea ce nu vedem nici aici, deși are reprezentanți remarcabili și la noi în țară, este* laboratorul de matematică. *După părerea noastră,* laboratorul de informatică, *în curs de edificare și înzestrare, ar trebui cuplat în același spațiu, sau în încăperi vecine și deschise una alteia, cu laboratorul de matematică.*

In finalul cap. 15 se discută metodica elaborării programelor, dîndu-se apoi un sumar al introducerii calculatoarelor în învățămîntul din unele țări.

Această problemă este reluată pe un plan superior conceptual și aplicativ în cap. 16 „Educație și informatică" reelaborat integral în 1990, cu includerea conceptelor esențiale din primul congres UNESCO „Educație și informatică", ce a avut loc la Paris la sfîrșitul lunii aprilie 1989. Sînt cuprinse de asemenea programe analitice pentru pregătirea elevilor și profesorilor din SUA, URSS, Franța, Japonia, soluțiile problemelor de la concursurile naționale românești din 1987—89, temele recomandate (și rezolvate) pentru cluburile de informatică. Capitolul are legături evidente cu textul PROLOG—DIALOG—EPILOG și continuări în ANEXE.

Partea a VIII-a *prezintă „Programe educaționale în BASIC, pe calculatorul HC 85".*

Găsim programe pentru clasele I—VIII folositoare în rezolvarea unor anumite clase de probleme, dar și altele adaptate unui singur tip de probleme, cum ar fi trasarea graficelor unor funcții elementare.

Programele pentru clasele IX—XII cuprind noțiuni delicate, ca funcția modul, grafice de funcții trigonometrice directe și inverse, funcții exponențiale și logaritmice, trasări de conice și de alte locuri geometrice. Trecîndu-se apoi la alte discipline, avem programe de teoria cinetico-moleculară, de optică, de motoare termice, de osciloscoape etc. Această parte are o extindere suficientă pentru a se constitui singură ca o broșură cu caracter tehnic de programare.

Partea a IX-a, *(capitolul 19), prezintă un caracter original, fiind însoțită de trei casete magnetice audio cu înregistrări de programe pentru calculatorul HC 85. Acestea au ca obiectiv învățarea interactivă a operării, a deprinderii de a lucra la tastatură, a instruirii în BASIC și LOGO, a proiectării circuitelor logice etc. Programele au fost realizate la catedra de calculatoare a Facultății de Automatică a Institutului Politehnic București, sub coordonarea-autorilor, prof. dr. Adrian Petrescu și șef de lucrări dr. Nicolae Țăpuș, de cîțiva studenți pînă în 1987, astăzi ingineri, citați în lucrare, fiecare pentru programul elaborat. La acestea se adaugă programul de prezentare a calculatorului personal HC 85, elaborat la Întreprinderea de Calculatoare Electronice București sub coordonarea directorului tehnic, ing.* Traian Mihu, *el însuși coautor al lucrării.*

Pe casete mai sînt înregistrate programe ale unor elevi din clasele VI—XII de la școli generale și licee din București, Arad, Buzău, Iași, realizate în cluburile de informatică ale liceelor, în unele cazuri cu aportul IPB, ITC, ICI, centre de calcul. Vedem, deci, cîtă creativitate poate stimula la copii cunoașterea și folosirea calculatorului personal, în speță a calculatorului HC 85.

Complementelor de matematică, indispensabile folosirii calculatorului le este consacrată **partea a X-a**. *În afară de cîteva noțiuni introductive privind siste-*

mele de numerație, cu punerea în evidență a operațiilor elementare în bazele binară, zecimală, octală și hexazecimală, strict legate de operațiile pe calculator, se prezintă operatorii logici cu proprietățile respective, funcțiile logice și problema deciziei, un accent deosebit fiind pus pe noțiunile de algebră booleeană. După o scurtă prezentare pe baza unor definiții generale, se trece la simbolurile și regulile algebrei și se evidențiază unele modele utile ale acesteia, ca cele din logica propozițiilor, din teoria mulțimilor, din teoria schemelor cu contacte și relee etc.

Partea a XI-a *revine unor „Complemente informatice" unde se prezintă succint instrucțiunile microprocesorului Z 80, apoi instrucțiunile de utilizare a codului mașină la calculatorul HC-85, cu ajutorul cărora se pot scrie unele programe eficiente, cu viteze convenabile de operare și cu ocuparea unui spațiu redus în memorie. În sfîrșit, calculatorul personal-profesional FELIX PC și familia IBM-PS/2 formează obiectul ultimului capitol, 25, primul dintre acestea fiind realizat în țara noastră cu un grad ridicat de integrare tehnologică, cu o structură compactă și un sistem de programare susceptibil de numeroase aplicații. Cît privește familia de calculatoare personale IBM-PS/2, acesteia i se face o prezentare pornind de la faptul că a fost produsă în prima parte a anului 1987, deci prezintă o noutate stimulatoare, de interes larg.*

Lucrarea este completată printr-o bibliografie care acoperă întreaga arie de cunoștințe prezentată în cele două volume. O serie de actualizări și completări importante (instrucțiunile interpretorului BETA BASIC — un BASIC mai performant, cu care cititorii și cei ce utilizează casetele pot lucra, continuări ale soluțiilor problemelor de la concursurile naționale sau ale rezolvării temelor propuse în cap. 16, cum și multe altele fac obiectul celor 6 ANEXE, de excepție. Cartea (inclusiv casetele) este rodul unei munci intense, desfășurate de întregul colectiv de autori cu competență și devotament.

Sfîrșitul consistent al materialului PROLOG—DIALOG—EPILOG este un „happy end," fiind consacrat calculatorului HC 85 extins și calculatorului HC-88, produse adaptate lucrului cu dischete și lucrului în rețele.

Se cuvine să menționăm aici aportul tuturor membrilor colectivului, care au reușit să ofere cititorilor o lectură atractivă într-un domeniu care, de obicei, se caracterizează printr-o tehnicitate obositoare. Credincioși titlului ales pentru lucrare, ei au lăsat impresia unei excursii agreabile într-un teritoriu plin de atracții, dar și de obstacole, în realitate conducînd pe cititor să-l cunoască și să-l fructifice. Este indiscutabil un merit deosebit faptul că lectura lucrării interesează nu numai pe începători, dar și pe specialiști, care au ce să rețină din ea, în special din ceea ce autorii au numit „... și nu doar atît".

Ceea ce surprinde este faptul că porțiunile realizate de fiecare dintre aceștia se sudează fără discontinuități, dînd lucrării o unitate remarcabilă, ceea ce pune în valoare coordonarea ei.

Reușita acestei lucrări dovedește că lucrul în echipă poate realiza performanțe pe care individual nu le-ar putea obține fiecare dintre membrii echipei, calitate pentru care întreg colectivul de autori merită felicitări deosebite. O mențiune specială se cuvine coordonatorului lucrării, profesorul A. Petrescu, ca și redactorului P. Zamfirescu, pentru unitatea și coeziunea întregii lucrări. Putem prevedea dispariția ei de pe piața editorială într-un timp prea scurt pentru ca cei interesați, în număr foarte mare, să aibă toți șansa de a o procura și felicităm Editura Tehnică pentru inițiativa de a o realiza.

ACADEMICIAN NICOLAE TEODORESCU

Cuprins general

Volumul 1

Studiu introductiv (Acad. N. Teodorescu) V
Cuprins general vol. 1 şi vol. 2 XI
PROLOG—DIALOG—EPILOG (continuă în vol. 2) XIII
Cuprins detaliat vol. 1 XXVIII
Partea I — **Calculatoare, microcalculatoare şi calculatoare personale în ţara noastră şi pe plan mondial** 1
Partea II — **Calculatoare numerice. Realizare fizică. Baze aritmetice şi logice** 37
Partea III — **Calculatorul personal HC-85. Structură, componente, operare, programare** 66
Partea IV — **Programarea în limbajul BASIC pe calculatorul HC-85** 103
Partea V — **Programarea în limbajul LOGO pe calculatorul HC-85** 175 —312

Volumul 2

Cuprins general vol. 1 şi vol 2 V
PROLOG — DIALOG — EPILOG — (continuare din volumul 1) VI
Cuprins detaliat vol. 2 XIV
Partea VI — **Microcalculatorul HC-85 în procese industriale. Pachete de programe de aplicaţii pentru HC-85** 1
Partea VII — **Calculatorul personal în învăţămînt şi educaţie** 44
Partea VIII — **Programe educaţionale în BASIC pe calculatorul HC-85** 111
Partea IX — **Microbiblioteca de programe pe casete** 182
Partea X — **Complemente matematice** 192
Partea XI — **Complemente informatice** 236
BIBLIOGRAFIE GENERALĂ, CONSULTATĂ ŞI RECOMANDATĂ 275
ANEXE — **1. Metodică algoritmi; 2. Programe educaţionale; 3. Interpretorul BETA BASIC; 4. Concursuri informatică** (probleme rezolvate) (continuă din vol. 1) **5. Probleme în pseudocod; 6. Soluţii ale temelor din 16.6** 279
PROLOG — DIALOG — EPILOG (continuare din vol. 2); **HC-85 extins, HC-88 disc,** reţea .... 320—360

În atenția cititorilor:

Cărțile pe care le consultați fac parte din primul tiraj al lucrării „ABC de calculatoare personale". Ele au fost tipărite parțial în decembrie 1989 și parțial în aprilie 1990. În primele 4 luni ale anului 1990 au fost elaborate cca 200 pagini noi, care completează sau înlocuiesc elaborările 1989. În acest fel, cărțile sînt pe deplin îmbunătățite și actualizate la nivelul primului trimestru al anului 1990, ținîndu-se seamă de documentația străină la zi, de realizările recente ale industriei românești, de noile structuri și programe ale învățămîntului, cum și de stadiul și perspectivele informatizării în lume și în țară.

Sîntem conștienți, însă, de două tipuri de inadvertențe:

a) au rămas unele greșeli de tipar, ușor detectabile;

b) nu am putut elimina absolut toate cuvintele de tipul: „pionier", „utecist", țări socialiste.

Mai menționăm încă odată înființarea Comisiei Naționale pentru Informatică, dependentă direct de Consiliul de Miniștrii; totodată ministerul nou înființat a fost redenumit Ministerul Industriei Electrotehnice și Electronice.

Vă promitem că în următoarele tiraje din 1990, aceste erori nu vor mai exista.

PROLOG — DIALOG — EPILOG

O școală în care profesorul nu învață și el, e o absurditate. Cred că am găsit un motto pentru școala mea. E vorba aceasta extraordinară...: „Nu se știe cine dă și cine primește".

CONSTANTIN NOICA

Cîte ceva despre calculatorul personal românesc

Calculatorul personal reprezintă una din importantele realizări ale industriei de tehnică de calcul din ultimii ani, cu un impact deosebit în numeroase domenii ale activității sociale, între care un loc important îl ocupă domeniul educației.

În contextul revoluției tehnico-științifice actuale _informația_ capătă un rol de prim ordin, reprezentînd o resursă cu o pondere mereu crescîndă în avuția națională a oricărei țări.

Este evident faptul că numeroase profesii actuale sau viitoare au și vor avea ca obiect informația. În așa-numita „sferă de activitate informațională" sînt deja cuprinși conducătorii de la toate nivelurile, oamenii de știință și specialiștii, funcționarii din instituții etc. Ponderea acestor categorii de oameni este în continuă creștere în toate țările industrializate.

Puternica proliferare a calculatoarelor personale _aduce o contribuție importantă la facilitarea accesului la mijloacele electronice de prelucrare a datelor a unor largi categorii de utilizatori: ingineri, economiști, tehnologi, fizicieni, chimiști, matematicieni, medici, proiectanți, profesori, muncitori, oameni de artă, agricultori, elevi ș. a._

Consecințele utilizării calculatorului personal sînt legate de accelerarea procesului de luare a deciziilor, pe baza unui volum mare de informații prelucrate în mod corespunzător, creșterea calității activității de proiectare de echipamente, procese și dispozitive, folosirea mai eficientă a resurselor materiale și umane la diverse niveluri, realizarea unor importante economii de materii prime și combustibil, creșterea calității produselor, perfecționarea și îmbunătățirea activităților din sfera serviciilor etc.

Nivelul „mediu" atins de industria noastră de tehnică de calcul a permis ca, pe baza proiectelor realizate de specialiști din învățămînt-cercetare și producție, să se treacă la _producția de serie a mai multor tipuri de calculatoare personale, bazate pe microprocesoare._

După cum este cunoscut, un calculator personal are gabarit și greutate reduse, este portabil, încorporează în structura sa _unul sau mai multe microprocesoare_, o memorie internă cu o capacitate suficient de mare (peste 64 Ko) și dispune de: _o tastatură alfanumerică_, un _sistem de afișare pe ecran TV sau ecran plat_ (cu cristale lichide sau plasmă), _o memorie externă pe casete magnetice, discuri flexibile sau rigide_, posedă unul sau mai multe _limbaje de programare de nivel înalt_ și are un _preț accesibil_ (deocamdată pentru colectivități moderate).

Industria noastră de tehnică de calcul produce calculatoare _personale_ și _personal-profesionale_ bazate pe _microprocesoare de 8 și 16 biți_.

În categoria calculatoarelor _personale_ au fost incluse cele care folosesc un microprocesor de 8 biți, o memorie internă cu capacitate de pînă la 64 Ko, un echipament de vizualizare de tip TV alb / negru sau color, o memorie externă avînd ca suport caseta magnetică (aMIC, Prae, HC-85, TIM-S, Cobra).

Categoria calculatoarelor _personal-profesionale_ este ilustrată de sistemele bazate pe microprocesoare de 8/16 biți, memorii interne cu capacitate de minimum 64 Ko, dispozitive de vizualizare de tip monitor-profesional, echipamente periferice performante, unități de discuri flexibile / rigide, imprimantă etc. (FELIX M118, HC-88, TPD-Junior, FELIX-PC, XT-Junior).

Limita între cele două categorii nu este fixă. Ea se modifică continuu, o dovadă simplă fiind calculatoarele HC-85 extins, cu extensii de disc flexibil, monitoare profesionale etc.

Revenind la calculatoarele personale, specialiștii noștri au ajuns la un consens luînd ca _model de referință calculatorul de tip Sinclair Spectrum_, larg răspîndit între utilizatorii de calculatoare personale din Marea Britanie și din alte țări.

Modelele realizate, HC-85[1], TIM-S[2], Cobra[3] şi, de curînd, CIP 4[4], JET 5[5], HC-88[6], emulează aceeaşi arhitectură, în sensul că operează cu porturi de intrare / ieşire, registre generale şi set de instrucţiuni identice, în condiţiile unor structuri fizice diferite.

Cîte ceva despre informatizarea învăţămîntului românesc

Privind domeniul învăţămîntului, necesitatea cunoaşterii şi folosirii tehnicii de calcul în economie, în cercetarea ştiinţifică şi tehnologică, în medicină..., pe scurt — în toate domeniile permeabile algoritmizării, a condus la „coborîrea„ unor noţiuni ştiinţifice din domeniul informaticii, de la nivelul cursurilor şi practicii universitare (sau cel al liceelor de specialitate), la nivelul învăţămîntului public general. Astfel, au apărut—**mai întîi în licee— cercurile** privind informatica şi calculatorul. Ele s-au extins repede în cadrul activităţilor extraşcolare din **gimnazii**, pătrunzînd apoi în viaţa unor şcoli primare şi chiar mai jos. Toate aceste preocupări s-au constituit mai întîi ca activităţi **facultative**, acolo unde perfecţionarea şi stăruinţa oamenilor s-au ridicat la nivelul cerinţelor unor astfel de preocupări. Pregătirea universitară şi, într-o anumită măsură, cea postuniversitară a personalului din învăţămînt, iniţiativa acestuia la nivelul şcolii, al inspectoratelor şcolare şi al Ministerului Învăţămîntului, sprijinul acordat de către facultăţile, întreprinderile şi instituţiile de profil informatic, toate acestea au făcut să crească interesul şcolii pentru informatică şi calculator.

Ca urmare, în anul şcolar 1987—88 au fost inserate, în cadrul programei de matematică a tuturor liceelor ţării, cîteva noţiuni de bază privind informatica şi programarea calculatoarelor. S-a resimţit însă **lipsa dotării cu tehnică de calcul a şcolilor, cabinetele de matematică sau laboratoarele şcolare pentru folosirea calculatorului ca mijloc de învăţămînt fiind destul de puţine şi, mai ales, destul de slab înzestrate.** Se simte nevoia unor întreprinderi care să asigure tehnica de calcul necesară şcolilor. Cu sprijinul Ministerului Învăţămîntului, Învăţămîntului Politehnic şi al celui Universitar, ori al unor licee devenite „pilot" în acest sens; cu sprijinul I.T.C.I. şi al filialelor sale, al centrelor teritoriale de calcul electronic, a fost lansată şi susţinută acţiunea de organizare a **taberelor** centrale sau judeţene de informatică pentru copii şi pentru profesori. Pe baza experienţei dobîndite s-a reuşit definitivarea de către Ministerul Învăţămîntului a unei **programe tematice anuale pentru taberele de vacanţă**, destinate pregătirii copiilor din învăţămîntul primar sau gimnazial în programarea şi utilizarea calculatoarelor. Pot fi astfel menţionate taberele de calculatoare organizate cu sprijinul Întreprinderii de Calculatoare Electronice, al Facultăţii de Automatică a I.P.B., al Institutului de Tehnică de Calcul şi Informatică, al unor facultăţi de profil din Cluj, Iaşi şi Timişoara, al unor centre de calcul şi licee din ţară. În acelaşi cadru se înscriu şi taberele locale, organizate la nivelul unor centre judeţene sau ministere, cu sprijinul unor licee în care s-au organizat laboratoare dispunînd de tehnica de calcul compatibilă procesului de învăţămînt. Cursurile acestor tabere de vacanţă se încheie prin lucrări teoretice şi practice, care stabilesc o ierarhie a competenţelor, stimulînd astfel gîndirea şi acţiunea creatoare a copiilor.

Stimularea şi evaluarea activităţii cercurilor de informatică din şcoli s-au realizat prin intermediul **sesiunilor ştiinţifice** la nivel de şcoală, interşcoli sau la nivel judeţean şi republican. Începînd cu anul şcolar 1985—86, pentru stimularea şi evaluarea activităţii cercurilor de informatică s-au introdus concursurile şcolare locale, judeţene şi republicane sub titlul

---

[1] HC-85 a fost proiectat ca model de laborator la Catedra de Calculatoare din IPB (prof. dr. ing. A. Petrescu şi asist. ing. F. Iacob) şi reproiectat tehnologic, ca model industrial, în vederea introducerii în fabricaţia de serie de ing. E. Dobrovie şi ing. S. Anghel de la ICE.

[2] TIM-S a fost proiectat de către specialiştii de la Catedra de Calculatoare din IPT (colectiv condus de prof. dr. ing. C. Strugaru) şi de la FMECTC Timişoara.

[3] Cobra este proiectat şi produs de Filiala ITCI Braşov (dr. ing. Gh. Toacşe şi colectiv).

[4] **CIP (Calculator de instruire programabil) proiectat şi produs de Într. Electronica, din 1989/90, cu tiraj de masă, cu interpretorul BASIC pe casetă şi cu un preţ de desfacere de 9850 lei.**

Menţionăm că în volumele 56 şi 57 din seria AMC a Editurii Tehnice, ce apar în trim. II 1990, este publicat limbajul BASIC pe CIP.

[5] **JET (Jocuri electronice pe televizor) este proiectat şi produs de Într. Electromagnetica din 1989/90, cu interpretor încorporat şi preţ în jur de 11 500 lei, pentru desfacere către public.**

[6] Spre deosebire de CIP şi JET (produse de larg consum) HC-88 proiectat la Întreprinderea de Calculatoare Electronice din 1989 (proiectanţi: ing. T. Mihu, ing. E. Dobrovie şi ing. V. Cososchi) este prevăzut cu unitate de disc flexibil şi asigură o dublă compatibilitate, HC-85 şi CUB-Z (CP/M).

„Informatica pentru utilizatori“. Această mişcare a cuprins, pe de o parte, reprezentanţii cercurilor din gimnazii, iar, pe de alta, pe cei din licee, exceptînd liceele sau clasele cu profil informatic, al căror concurs a fost organizat cu mai bine de 15 ani în urmă.

În cadrul tuturor acestor concursuri ale inteligenţei şi muncii s-au evidenţiat mulţi dintre cei mai talentaţi elevi, programele lor devenind în unele cazuri **produse-program pentru biblioteca şcolară**. Este, desigur, necesară stimularea activităţii şcolilor pentru crearea de produse-program necesare procesului de învăţămînt şi **înfiinţarea unor instituţii pentru realizarea şi sprijinirea realizării acestor programe la nivelul parametrilor de performanţă didactică mondială**.

Ca o confirmare a acestor preocupări şi a eficienţei activităţii de instruire în cercurile de informatică, vom evidenţia rezultatul obţinut de echipa României la Concursul Internaţional de Programare* de la Sofia, Bulgaria, din perioada 17—20 mai 1987, în cadrul celei de-a II-a Conferinţă internaţională „Copiii în era informaticii — facilităţi pentru creativitate, inovaţie şi noi activităţi“.

În acelaşi cadru se înscriu şi taberele de calculatoare organizate anual, în timpul vacanţei de vară pentru studenţi, precum şi cele pentru profesorii din gimnazii sau din licee. Menţionăm aici faptul că, pe baza unor cursuri de cîte zece zile, organizate consecutiv, de-a lungul a trei vacanţe şcolare, Liceul „Dimitrie Cantemir“ din Bucureşti a continibuit la iniţierea în domeniul cunoaşterii şi folosirii calculatorului în procesul de învăţămînt de către profesorii de matematică şi fizică de la liceele patronate de Ministerul Industriei Uşoare. Această activitate de sprijin a fost extinsă, prin colaborare cu unele ministere, care au organizat astfel de cursuri pentru profesorii de matematică, fizică, chimie şi biologie.

Ca urmare a acestor preocupări, impulsionate şi de introducerea, în anul şcolar 1987—88, în **programa de matematică** a claselor, IX—XII, a unor **noţiuni de informatică şi programarea calculatoarelor**, s-a accentuat cerinţa dezvoltării producţiei de calculatoare personale, a documentaţiei tehnice şi a cărţilor de informatică educaţionale. De bun augur, în acest sens, au fost şi manualele privind tehnica de calcul şi informatica — existente prin planul de învăţămînt al liceelor de matematică şi fizică, sau pe profilul pur informatic. Cartea de faţă încearcă să răspundă şi ea acestui deziderat. Ea s-a constituit ca rod al colaborării mai multor specialişti din învăţămînt şi cercetare, care au participat efectiv la proiectarea calculatoarelor personale, în introducerea în fabricaţie şi la experimentarea acestora în cadrul cercurilor de elevi din şcoli, la organizarea şi îndrumarea taberelor de calculatoare şi informatică, la reciclarea unor cadre didactice şi la dezvoltarea prin publicaţii a schimbului de idei şi de experienţă. În consecinţă, în această carte au fost incluse şi cîteva din contribuţiile unor elevi care s-au distins la sesiunile de comunicări ale elevilor, la concursuri, sau în cadrul cercurilor de informatică ale unor şcoli.

Lucrarea de faţă apare în cadrul Editurii Tehnice, avînd şi scopul de a se constitui ca **ghid pentru activităţile privind informatica din şcoli**. De remarcat că prin restructurarea sa, la iniţiativa redacţiei de specialitate, cadrul iniţial al acestei lucrări a fost lărgit (ca şi colectivul său) iar cartea rezultată îşi lărgeşte aria cititorilor cărora se adresează. Ea are astfel un caracter de masă, multe dintre noţiunile introduse şi tratate în diversele sale capitole fiind accesibile şi copiilor în clasele primare. Unele capitole din lucrare presupun cunoaşterea unor noţiuni de bază din domenii ale fizicii şi chimiei, ale electronicii şi electrotehnicii — fie la nivelul claselor a VII-a şi a VIII-a, ori al claselor a IX-a şi a X-a, fie la nivelul avansat al claselor a XI-a şi a XII-a. **Prin mersul ei, de a simplu la complex, lucrarea se doreşte a fi accesibilă oricărui om care doreşte să se iniţieze în informatică şi în aplicaţiile acesteia.** În acelaşi timp, cartea încearcă să fie şi **ghid pentru profesor, un îndrumător metodologic şi pedagogic pentru toţi cei care conduc cercuri de elevi în domeniul informaticii**, un material de pornire pentru membrii cercurilor de matematică şi informatică din şcoli, pentru cercurile şcolare ale altor discipline teoretice sau tehnice, care folosesc calculatorul în munca lor ştiinţifică. Ea devine deci **un material auxiliar învăţămîntului** din şcoli, mai ales în munca la clasă şi în conducerea cercurilor din gimanzii şi din licee. Multe din capitolele acestei lucrări se pot folosi ca teme pentru cercurile de elevi. În acest scop se pot cita, spre exemplu, capitolele 21, 21 şi 22, ele fiind realizate pe baza lecţiilor de iniţiere în calcul numeric, logică şi algebra booleană, folosite la cercurile de matematică şi informatică de la Liceul „Dimitrie Cantemir“ din Bucureşti, în perioada 1968—1988. Evident, materialul destul de succint prezentat în capitolele enunţate mai sus nu are pretenţii de originalitate, unele din noţiunile şi exemplele expuse fiind preluate din prestigioase manuale şi cărţi şcolare sau universitare,

* Au participat 7 echipe din 6 ţări: R.F.G., România, Bulgaria, Cehoslovacia, Ungaria, U.R.S.S. — aceasta fiind ordinea descrescîndă a clasamentului pe echipe. Elevul Răzvan Jigorea, din Arad, a obţinut premiul special al Juriului.

conform bibliografiei citate, ele fiind în parte accesibile ultimelor clase de gimnaziu şi îndeosebi primelor două clase de liceu.

Există în lucrare şi unele capitole care sînt prezentate în detaliu şi pentru prima dată în cărţile noastre de specialitate. Amintim în acest sens limbajul LOGO (capitolele 10, 11 şi 12). Se poate menţiona, de asemenea, prezenţa alături de carte a primelor casete magnetice cu programe care pot fi folosite pentru învăţarea operării calculatorului HC-85, a limbajului BASIC, a proiectării cu circuite logice combinaţionale şi secvenţiale etc. Casetele mai conţin interpretorul pentru limbajul LOGO, aplicaţii LOGO şi o serie de programe pentru clasele V—VI, respectiv IX—XII, şi pentru cercurile de elevi.

Desigur, există multe elemente de noutate pentru tineretul şcolar în această lucrare, dar să lăsăm în seama cititorului această problemă. Am menţiona totuşi aici că dacă această lucrare ar fi putut să apară în anii 1987—88, cînd fusese pregătită pentru tipar, caracterul său de noutate şi, mai ales, de utilitate ar fi cîştigat mai mult.

Aşa cum s-a mai spus în interiorul acestei cărţi, calculatoarele sînt folosite în procesul de învăţămînt, în cercurile pentru pregătirea superioară a elevilor şi în laboratoare pentru diverse tehnologii, fiind cuplate cu echipamente de laborator, instrumente ştiinţifice, miniroboţi industriali etc. Toate acestea permit pregătirea elevilor la nivelul cerinţelor solicitate de necesităţile dezvoltării viitoare a ţării noastre şi ale cooperării libere şi eficiente cu toate statele lumii. Pentru aceasta va trebui să facem loc unor largi iniţiative în domeniul folosirii informaticii la vîrste foarte mici, prin jocuri accesibile şi atractive pentru preşcolari sau pentru şcolarii din clasele primare.

Tocmai unui astfel de scop îi serveşte, spre exemplu, o parte din programul Cercului experimental-pilot, numit MINICOMP, care funcţionează în cadrul Institutului de Cercetări în Informatică Bucureşti. Condus de specialişti din Institut, cercul are ca scop elaborarea şi validarea celor mai potrivite metode de instruire a elevilor, în vederea generalizării unor metodologii la nivelul cercurilor de acest tip din întreaga ţară. Se organizează anual două tabere de pregătire (iarna şi vara), acestea cuprinzînd peste 200 de copii şi 50 de profesori din ţară. La aceste tabere îşi aduc o importantă contribuţie profesorii din învăţămîntul general şi din învăţămîntul superior (catedra de calculatoare din Institutul Politehnic Bucureşti), specialişti din institut, de la Fabrica de calculatoare electronice şi de la Centrele de calcul.

În prezent funcţionează sute de cercuri de informatică, în care activează zeci de mii de elevi. Se creează astfel un larg acces la baza de software existentă pentru calculatoarele personale cu aceeaşi arhitectură.

Cîte ceva despre educaţie şi informatică în lume pînă în 1990

Deşi tehnicile şi strategiile pentru introducerea calculatoarelor şi a tehnologiilor informaţionale în învăţămîntul din diverse ţări* sînt diferite, există tendinţe comune.

Tehnicile naţionale, regionale, locale răspund — în măsuri neegale — aceloraşi presiuni externe:

— **cereri de restructurare a economiei;**

— **interese industriale** (introducerea tehnologiei informaticii în şcoli completează şi întăreşte dezvoltarea industriei naţionale microelectronice);

— **cerinţe comerciale** (fabricaţia promovează introducerea unor tehnologii în şcoli ca parte a propriilor strategii de marketing, de creştere a vînzărilor către diferite unităţi şi către familii);

— **dorinţe ale părinţilor şi copiilor;**

— **concepţii culturale** favorabile asocierii la tehnologii informaţionale electronice;

— **tendinţe ale unor lideri politici** de a demonstra îmbunătăţiri vizibile şi concrete ale sistemului educaţional;

— **dezvoltări tehnologice**, care influenţează, ele însele, dimensiunea şi viteza de implementare, cu reducerea concomitentă a costurilor.

Ca răspuns la aceste presiuni, multe dintre ţările dezvoltate au iniţiat la începutul anilor '80 — programe ambiţioase şi costisitoare.

Către mijlocul anilor '80 s-au desprins două grupuri de ţări:

Primul grup, care a dezvoltat strategii „**restrictive** cu două obiective: 1 — introducerea învăţării ştiinţei calculatoarelor în clasele secundare superioare şi în şcolile de specialitate

---

* Pierre Duguet, administrator şef al CERI din OECD „National Strategres and Their Extension to the International Level", UNESCO, Paris, 1989.

* Fourth Conference of Ministers of Education of Member States of the Europe Region „Informatics in education", UNESCO, sept. 1988.

pentru a dezvolta noi perspective în piața muncii și 2 — alfabetizarea în calculatoare la toate celelalte nivele educaționale. Austria, Belgia, Danemarca, Finlanda, R.F. Germania, Grecia, Irlanda, Italia, Japonia, Olanda, Norvegia, Suedia, Elveția și multe state ale U.S.A. aparțin acestui grup.

— **Al doilea grup** include țările cu strategii **de mai largă înțelegere**, deci nu numai de a promova instruirea în știința calculatoarelor sau alfabetizarea informatică ci, în principal, și de a utiliza tehnologiile informaticii și ale comunicațiilor, pentru a îmbunătăți procesele de instruire și învățare; (sînt incluse obiective complementare, ca: îmbunătățirea calității procesului de instruire, mai buna cunoaștere și influențare a proceselor de învățare, reconstruirea programelor analitice, compensarea scăderii în număr și calitate a profesorilor, îmbunătățirea accesului la educație, dezvoltarea comunicațiilor între școli, pe plan intern și internațional. Exemple sînt Franța, Marea Britanie și ținutul Ontario din Canada.

Există tendința de trecere către **strategii comprehensive mai largi**: Italia (Programul 1985), Portugalia (Planul Minerva, 1984—1988) și Spania (Planul 1985—1989), sau chiar de planuri radical schimbate: Olanda (Planul OSTA, 1989—92). În S.U.A., un recent raport al Oficiului de mijloace tehnologice al Congresului American relatează că 24 state au planuri de lungă durată, iar 13 state au dezvoltat asemenea strategii (Power On! New Tools for Teaching and Learning, September 1988). În Japonia, Centrul de educație în calculatoare a stabilit, în 1986, ca MITI și Mombusho să investigheze sistemele educaționale bazate pe calculatoare. Desigur, există însemnate diferențe între intensitățile acestor procese în diferitele țări menționate, legate de resurse, de opțiunile de a modifica sau nu metodele tradiționale de organizare școlară etc. De asemenea, se evidențiază două principale strategii de implementare:

1. **De la școli pilot** selectate, **către toate școlile** (strategii influențate de resurse limitate și de evaluări ale eficienței pentru a atinge o „masă critică" de acces la hardware, la o integrare a calității software-ului cu programele analitice și la o pregătire corespunzătoare a profesorilor; școlile ce aplică aceste metode sînt de ordinul zecilor în Norvegia, de ordinul-sutelor în Olanda, Portugalia, Spania; Suedia a lansat un plan „Action programme for computer science education in schools, adult education and teacher training, (1988—1992)", care include experimentarea învățării asistate de calculator într-un număr limitat de clase (250) din învățămîntul general. Este un contrast cu puține țări, ca Franța și Marea Britanie, unde școlile sînt toate echipate, chiar dacă numai cu un mic număr de calculatoare; aceste strategii sînt alese din rațiuni politice, pentru înlăturarea inegalităților între școli, pentru stimularea inițiativelor individuale ale profesorilor, elevilor, directorilor și părinților.

Strategia „ideală" este de a **combina aceste două direcții** către generalizarea dotării cu calculatoare (Japonia, Suedia și unele state din S.U.A. — exemplu de vîrf fiind Florida, cu raportul 1/18 calculator / elev și un program de 10,5 mil. $ pentru „5 Model Technology Schools").

2. De **la centralizare către descentralizare**, care reflectă rolul inportant al asistenței la nivelul național (ca în Franța, cu învățămînt centralizat) sau local în Norvegia (și în statele federale cu învățămînt descentralizat). Statele au rol extins în 5 domenii: **hardware** — cu compatibilități soft facilitate de aceleași sisteme de operare; **software** — organizarea unei piețe viabile și informarea profesorilor cu realizările învățămîntului asistat; **pregătirea profesorilor**, în timpul sau în afara serviciului; **cercetarea** — coordonată și susținută financiar; **evaluarea** strategiilor centrale,

Este posibil că **descentralizarea** asigură o dezvoltare puternică a creativității, **cum** rezultă din multe experimentări, care arată că **abordarea strictă top-down este o tendință perturbatoare**.

Țările aplică și tehnologii specifice:

— În **hardware** — multe țări au decis să dea prioritate sau să impună utilizarea calculatoarelor de fabricație națională: Ontario — Canada (Icon); Franța (Thomson, Bull, Goupil); Italia (Olivetti); Olanda (Philips, Compudata); Norvegia (Scandis, Tiki), Portulagia (Enner 1000), Spania (Computer XP), Suedia (Compis, Microber, ABC), Marea Britanie (BBC, RM și Sinclair), Statele Unite (Apple, TRS, Commodore); Iugoslavia (Iskra, Orel). Această politică promovează industria microelectronică proprie, nu numai în piața școlară ci și în piața familială, în țară și în afara ei, iar, pe de altă parte, favorizează reducerea incompatibilității hardware, problemă acută în multe școli (În Japonia în '87 s-au găsit 100 tipuri de calculatoare — iar în 1988, 11 companii electronice au realizat un prototip de calculator educațional bazat pe sistemul de operare TROM; în Norvegia, în 1982 — 75 tipuri). Un număr de țări au autorizat achiziționarea de calculatoare străine, în general compatibile cu sistemul de operare MSDOS sau cu mediul Unix.

O altă tendință este creșterea numărului de calculatoare (1 micro la 40 elevi în învățămîntul secundar din țări industrializate) și creșterea puterii mașinilor, corelată cu noile pachete puternice de software educațional sau profesional (procesor de texte, tabelării electronice, baze de date). Aceste aspecte țin de experiența țărilor celor mai avansate, depinzînd strict de poziția națională.

— **În software**, cerința principală este integrarea cu programele analitice a unui software de înaltă calitate, care — însă — costă mult. Este un fel de cerc vicios: software-ul de calitate rămîne scump dacă piața este îngustă, iar piața rămîne îngustă dacă avem prețuri mari. Statul poate stimula piața, încurajînd producătorii de software și reducînd, astfel, costurile. Se estimează că cea mai mare parte a software-ului educațional este de tip instruire programată simplă, doar 10—20% fiind proiectat pentru a apela la procese de gîndire de ordin mai înalt. Oricum, în S.U.A. există disponibile peste 10 000 de programe de instruire, iar în Australia, Canada, Franța, Italia sau în Marea Britanie — între 1 000—4 000. Aceasta este situația în țările cu piețe mari. În ce privește cooperarea cu țările mici, dificultățile țin de diferențele între obiectivele educaționale și, în consecință, de programele analitice; cooperarea ar putea avea loc la diferite nivele: la un anumit nivel se poate opta pentru un transfer global al pachetelor de programe între țări similare; la alt nivel, țările pot coopera în proiectarea unui software apecific, pe baza unei gîndiri pedagogice comune (de ex. S.U.A. și unele provincii canadiene, țările scandinave, unele țări din piața comună, sau alte țări care ar putea găsi sprijin pentru realizarea unor produse de instruire interactive, evitîndu-se „reinventarea roții").

Țările cele mai avansate se confruntă și cu problema **informării profesorilor** asupra **calității produselor software** disponibile (corespondența cu cerințele programelor analitice, conținutul și metodologia de lucru, adresanții, prețul, ce hardware și software este necesar pentru funcționare, cum a fost evaluat și de cine). Pentru a asigura aceste informații și pentru a evita respingerea unor produse neadecvate, achiziționate pe bază de reclame comerciale (și — ca o consecință — rejectarea calculatorului ca asistent), în multe țări au apărut agenții guvernamentale însărcinate să revizuiască și să evalueze software-ul existent și să acorde „etichete de calitate" pentru cele mai multe produse. Este cazul Australiei, Canadei, Franței, al Japoniei, Marii Britanii, S.U.A (la nivelul statelor de ex. California sau în Florida). Se constituie baze de date, cu acces internațional.

Pregătirea profesorilor este o problemă crucială; situația nu este deosebit de bună, existînd, în afară de deficitul de hardware și software și o lipsă de profesori, cu pregătire mai ales după serviciu și, mai puțin prin sistemul de pregătire anticipată. Multe state fac eforturi în această direcție, **la două nivele**: pe de o parte prin cursuri introductive (una sau două săptămîni de familiarizare cu tehnologiile informatice și cu operarea echipamentelor în clasele lor; în Franța cca 25% din profesori, iar în S.U.A. circa 30%, au fost „alfabetizați" (2—5%, în țări în curs de dezvoltare); pentru unele categorii de profesori, la nivele școlare specifice, situația este mai bună, astfel, cca 95% din profesorii danezi din școlile secundare superioare și își completează „alfabetizarea" cu cca 20 ore suplimentare în școlile lor; de asemenea, utilizarea benzilor video pentru completarea cunoștințelor este răspîndită în unele țări, ca de ex. în Marea Britanie.

Dar, această pregătire introductivă poate fi insuficientă și se pune problema unui **nivel avansat**: astfel, în Franța, există programe cu durata de pregătire de pînă la un an, prin care se „antrenează echipe limitate de profesori" care apoi, în „cascadă" pregătesc alte grupe de colegi. Sistemul este costisitor și nu este atît de eficient, datorită și diferențelor între echipamentele pe care profesorii le vor folosi în școlile lor: o tendință nouă (de ex. în Portugalia) este pregătirea profesorilor la serviciile lor și acordarea lor a unui statut de „cercetători" (uneori în cooperare cu universități și cu alți colegi) asupra pachetelor de programe ce le utilizează.

În 1984, Organizația pentru Cooperare Economică și Dezvoltare (OECD) a organizat o plenară a 122 universități și institute de cercetare din 9 țări membre, dedicată **cercetării și evaluării** domeniului informaticii educaționale. Cercetarea este îndreptată spre rațiunile pedagogice, spre cele politice, pentru calificarea eficienței reale a tehnologiilor informaționale în procesul de învățare. Evaluări ale informatizării au avut loc și în Anglia (1987), Australia (1986), Scoția (1987), Franța (1986), Norvegia (1987), S.U.A. (1988), unde s-au publicat rapoarte importante, care reflectă experiența școlilor pilot și care evidențiază „masa critică" a accesului la calculatoare, calitatea software-ului integrat în programele analitice și pregătirea profesorilor pentru utilizarea pedagogică a calculatoarelor. În Marea Britanie, o unitate specială recentă a Departamentului de Educație și Știință supervizează cercetările principale orientate asupra eficacității utilizării tehnologiilor informaționale în procesele de instruire și învățare.

Este important, de asemenea, pentru organismele guvernamentale și locale ca, în perspectiva spre anul 2000, în condițiile actuale, cînd tehnologiile nu mai constituie probleme, nici o carte din domeniu să nu mai apară fără un set de dischete, cu strategii educaționale complementare conținutului cărții, cu o mare cantitate de exemple, cu metode de instruire programată, cu concepte interactive de explorare și testare a ipotezelor, cu metode de simulare, cu baze de date relevante etc.: pentru **pregătirea unor sisteme educaționale cît mai consistente**, în cît mai scurt timp, posibil, pentru a face față schimbărilor inevitabile pe care le așteptăm în **învățămîntul informatizat**.

Se prevede ca numărul microcalculatoarelor în școlile din Europa de vest să fie în 1990 de trei milioane, crescînd în ultimii trei ani cu 2 milioane. (În 1986 erau 1 milion în școli, în afara celor 12 milioane calculatoare personale aflate în funcțiune în Europa de vest — din care 27,6% în Franța, 25,3% în Marea Britanie, 13,8% în R.F.G., 11,3% în Italia / Spania, 9,2% în Scandinavia, 4,7% în Olanda, 8,3% în alte țări. În 1990 se prevede 20% în Franța, 18,2% în R.F.G., 17,9% în Marea Britanie, 19,4% în Italia / Spania, 8,7% în Scandinavia, 5,9% în Olanda, 9,9% în alte țări — din totalul de 3 milioane unități).

Sectorul cel mai avansat al învățămîntului este cel al **învățămîntului superior**, cele mai multe universități vestice oferind tuturor studenților puterea calculatoarelor, iar 70% din studenți lucrînd cu computere și acasă: și universitățile din est anunță că au generalizat introducerea științei calculatoarelor. **Școlile secundare** sînt, de asemenea, sub „vizorul" multor țări, cu o atenție aparte asupra **ciclului 2**, liceal, al dezvoltării software-ului și al pregătirii profesorilor: calculatoarele sînt materie de învățămînt, uneori opțională, în învățămîntul general, și în cel specializat; în R.F.G. sau Italia firmele de calculatoare sînt asociate la instruire, îndeosebi pentru școlile specializate. Calculatoarele sînt utilizate și drept instrumente de instruire, în forma unor pachete tutoriale sau de simulare, sau în forma unor programe utilitare produse de industrie (prelucrarea textelor, tabelărie electronică, baze de date, grafică interactivă), sau a unor programe specializate pe subiecte școlare. În primul **ciclu al școlilor secundare** programele sînt mult mai modeste, în simularea limbajelor, în dezvoltarea conceptelor, mai ales în matematică. În ce privește **învățămîntul primar**, numai puține țări (Marea Britanie, Franța) au introdus computerele, în mod special pe baza limbajului LOGO. Alte țări (Belgia, Italia, Danemarca, Olanda, Iugoslavia) s-au limitat la experimente, iar în alte părți (ca în unele landuri ale R.F.G.) implementarea calculatoarelor a fost limitată din motive medicale; ele se utilizează doar pentru jocuri logice și pentru exerciții în forme scrise; puține țări sînt interesate la nivelul **învățămîntului preșcolar** și numai pentru inițierea în organizarea spațiului și a conceptelor geometrice.

Multe țări din Vest, ca și din Est, au încurajat înființarea unor **cluburi** în afara orelor de școală, **deschise unor comunități mai largi**, pe bază voluntară, principalele activități fiind **scrierea și schimbarea** programelor. Pentru **educația handicapaților** sînt preocupări, în deosebi pentru programe Braille și pentru copiii care învață mai greu.

Educația la distanță progresează odată cu dezvoltarea rețelelor de calculatoare, care permit accesul la bănci de informații, la stații de instruire la domiciliu, la videotext și la poșta electronică, la transmisii digitale, la diversificarea unor instrumente interactive pentru instruirea personalizată și în locuințe. Spre exemplu, Austria a organizat un sistem de lecții pe calculatoare personale pentru studenții care dispun **de terminale la locuințe**.

În ultimii 20 de ani **consumul de putere de calcul** a crescut după o curbă exponențială cu o dublare la fiecare an; astăzi, cea mai mare parte din aplicații se realizează pe echipamente de **„birou"** (desktop computer). De la Apple II (1978) și de la IBM PC (1981), cu diplay-uri de mică rezoluție și memorie limitată, la modelele noi suprarapide, cu diferite sisteme de operare, cu memorii mari și grafică superioară; apare o întrebare legitimă, cît timp calculatoarele personale Macintosh (1982) și IBM PS/2 (1986) vor mai rezista, chiar cu marea lor varietate de standarde grafice, în fața așa-numitelor **stații de lucru?**; acestea sînt caracterizate prin mai multe milioane de instrucțiuni pe secundă (MIPS) ale microprocesorului — inima unității centrale (CPU), printr-o memorie principală de 2—8 milioane de caractere (Mbytes) și un ecran depășind un milion de pixeli color. **Discurile** au sute de Mbytes, ca memorii externe și mai mulți Gigabytes în rețele (memorii optice pot fi incluse). **Sistemele de operare** sînt de tip multiproces, alocînd mai multe ferestre, care operează simultan pe ecran: software-ul avansat implică grafică interactivă și animație: conectarea la **rețele locale** (LAN) și **spațiale** (WAN) permite dezvoltarea poștei electronice și a altor sisteme similare.

Toate acestea nu puteau să nu influențeze informatizarea învățămîntului, ca și progresele în inteligența artificială în traducerile automate, recunoașterea vorbirii și formelor, sistemelor expert ș.a. Producătorii de hardware ca și editorii nu au investit mult în software-ul educațional spre deosebire de companiile de software.

Cartea de față poate constitui, prin întregul său conținut, un suport pentru o serie ierarhică de manuale de tehnică de calcul și informatică. Cuprinsul său poate orienta în elaborarea unor noi programe analitice românești, începînd din 1990. În același scop, am prezentat în capitolul 16 programe analitice de informatică pentru școli primare, școli secundare, „alfabetizare" informatică, pregătirea profesorilor, din diverse țări (S.U.A., Franța U.R.S.S., Japonia). Aici, am găsit necesar să redăm și să comentăm — pentru a avea un tablou mai complet — și o programă analitică pentru integrarea informaticii în studiile tehnico-inginerești din școli speciale și superioare din R.F.G..* Este prevăzută pentru **cursuri de 3 trimestre**, în primii doi ani de studiu. Informatica este considerată o metodologie științifică pentru tehnologia informației, ce interesează pe ingineri și tehnicieni ca instrument de proiectare, modelare, simulare, gestiune, fabricare, calcul.

Programa poate fi divizată în trei nivele: 1) **utilizarea calculatoarelor cu software de aplicații standard**; 2) **aplicarea optimă a informaticii în procesele inginerești**; 3) — **dezvoltarea programelor și configurațiilor hardware pentru sarcini inginerești.**

Obiectivele globale ale programei sînt: 1° — cunoașterea echipamentelor (componente, operare, instalare, diagnoza erorilor); 2° — cunoașterea sistemelor de operare și a utilitarelor (funcțiuni, comenzi DOS, rutine pentru conducere date, fișiere ș.a.); 3° — introducerea înprograme aplicative standard, prelucrare texte, baze de date, tabelărie electronică, grafică, software de comunicații); 4° — conceptele științei calculatoarelor (informația ca obiect manpulabil, reprezentarea informației prin date memorate, redarea datelor în forme textuale sau vizuale, bazale înțelegerii viitoarelor domenii ale informaticii, mijloace de învățare a instrucțiunilor; termenii de CAD, CAM, sisteme specializate, procesare paralelă, transputere); 5° — desvoltări de software (structuri de programe, structuri de control; structuri de date, scule de programare — editor, interpretor, compiler, linkeditor, generator, limbaje de programare — Turbo-Pascal, organizarea colecțiilor mari de date; metode de dezvoltarea software-ului — incluzînd testarea și documentarea —. proiectarea interfețelor); 7° — impact social și economic al tehnologiei calculatoarelor.

C u r s u l p r e l i m i n a r. **Calculatoarele personale drept mijloc de scriere și desenare** (semestrul I, 30 ore curs + 20 ore practică, un C.P. la 2 studenți; 1 CP cu disc rigid pentru profesor, conectat cu celelalte pentru încărcarea programelor, tablă, proiector cu tabletă CP; sesiunile **teoretică** și **practică**, ca blocuri separate; 10 studenți la un profesor). **Structură**: 1° — Explicarea unităților vizibile ale CP. (Unitate centrală, monitor, tastatură, memorie, discuri flexibile și rigide, imprimantă; încărcarea unui program standard, copii pe disc, întreținerea fișierelor, tipărirea fișierelor). 2° — Încărcarea și pornirea unui program de prelucrare de texte. 3° — Funcțiuni de bază ale procesării de texte — intrări, tipăriri, memorări — (editare, mișcare blocuri, construcție blocuri standard, funcții de căutare, redare texte și tabele). 4° — Întreținerea fișierelor, manipularea lor. 5° — Cunoașterea mai adîncită a funcțiunilor procesoarelor de texte. 6° — Cunoașterea mai adîncită a funcțiilor sistemelor de operare. 7° — Explicarea unităților funcționale ale CP. 8° — Practica instalării unităților și programelor, reconfigurarea lor. **Obiective**: operarea cu dispozitivele de intrare (tastaturi, șoricel...). Deprinderea executării cu PC a procesării de texte și de grafică tip comercial. Instalarea, după manuale, a hardware-ului și software-ului.

S e m e s t r u l II. **Fundamentele informaticii (I) pentru ingineri și tehnicieni** (60 ore, în clase cu aceleași condiții ca în primul semestru, plus programe CAD, bază de date relaționale, tabelărie electronică pe discuri rigide). **Structură**: 1° — Funcții de bază ale programelor CAD (desenarea unui obiect simplu tridimensional, adăugarea de texte sau măsurători). 2° — Analiza imaginilor, în termenii conceptelor informatice (limbajul vorbit, scris, simboluri, imagini, reprezentarea vizuală și percepția imaginilor, modele conceptuale de reprezentare a imaginilor) 3° — Opțiuni tehnice de stocare a datelor (caracter, înregistrare, fișier). 4° — Cunoaștere mai amînunțită a programelor CAD. 5° — Sisteme de baze de date (funcții de interogare, termeni de căutare, comenzi, analiza limitărilor). 6° — Abstracții ale funcțiilor principale, pe baza teoriei mulțimilor și a algebrei Boole. 7° — Organizarea datelor ca obiecte în baze de date relaționale. 8° — Mijloace de acces la bazele de date, limbaje de regăsire. 9° — Sisteme specializate, de ex. tabelărie electronică.

Obiective: lucrul cu programe CAD, de desenare, de organizare a datelor ș.a.

S e m e s t r u l III. **Fundamentele informaticii (II) pentru ingineri și tehnicieni** (60 ore + 40 ore practică, în blocuri, cu aceleași caracteristici, limbajul PASCAL).

---

* Peter Gorny, *Curriculum Proposal for the Integration of Informatics into Engineering Programs offered at Colleges and Universities in Developing Countries*, Congres UNESCO, Paris, aprilie, 1989.

Structură. 1° — Ce face un program în calculator (calculatorul — o maşină incompletă, analiza unui program simplu PASCAL de la program la codificarea în limbaj maşină, funcţiunile compilator lui PASCAL. 2° — Bazele limbajului PASCAL. 3° — De la problemă, prin algoritm, la program (descripţie semiformală prin pseudocod sau structograme, convertirea în programe, recunoaştere erori, specificare, proiectare, implementare, testare programe. 4° — Tipuri şi structuri de date (predefinite şi autodefinite, masive, înregistrări, fişiere). 5° — Subprograme modulare (aplicarea procedurilor şi funcţiunilor, declaraţii, accese la fişiere în programe PASCAL), interferenţe între programe Pascal şi alte programe. 6° — Metode simple de dezvoltarea software-ului (module, maşini abstracte, nume, interfeţe, parametru, abstractizarea datelor, modele shell, comunicaţii şi dialoguri între maşini, metode de documentare, practica programării sarcinilor inginereşti. **Conţinut şi metodologie**: calculatorul drept maşină cu stări finite, dezvoltarea sistematică a programelor, deprinderea lucrului în echipă pentru dezvoltarea soluţiilor din aplicaţiile inginereşti.

Se mai poate vorbi de un **semestru IV** (opţional) referitor, de exemplu, la hardware (arhitectură, proiectarea interfeţelor, reţelelor), la software (programare logică, grafică de computer).

Tot în perspectiva programului de informatizare a învăţămîntului ce se va derula cu siguranţă acum — după Revoluţia din decembrie 1989 — trebuie cunoscute răspunsurile date de 43 state membre ale UNESCO la chestionarele ce au precedat Congresul internaţional UNESCO **„Educaţia şi informatica"**, din aprilie 1989, Paris.

Astfel, **ierarhizarea obiectivelor statale prioritare pentru introducerea informaticii în educaţie** este: pregătirea tinerei generaţii pentru o nouă lume — 76 %; îmbunătăţirea procesului de învăţămînt — 54 %; ridicarea nivelului dezvoltării economico-sociale — 38 %; pregătirea specialiştilor — 33 %; pregătirea utilizatorilor — 30 %; descreşterea decalajului tehnologic între naţiuni —25 %, sprijin faţă de întîrzierile şcolare — 14 %; reducerea disparităţilor în cunoaştere dintre grupuri sociale — 11 %; facilitarea autoinstruirii — 8 %; facilitarea integrării sociale a celor fără lucru — 5 %.

Numărul calculatoarelor educaţionale varia, în diferitele ţări, de la zeci, sute, mii, sute de mii; în ţările pieţei comune erau 1 milion de calculatoare personale în 1988 şi mai mult de 1 milion în S.U.A.; se ajungea la 1 calculator pentru cel mult 50 copii.

Dificultăţile introducerii informaticii sînt de trei categorii: **Administrative**: (lipsuri strategice — 22 %, lipsa de suport ierarhic 19 %; limitări financiare — 60 %). **Tehnice**: echipament insuficient, incompatibil — 40 %; software insuficient sau de proastă calitate — 55 %; probleme de limbaj — 10 %; întreţinere — 20 %, electricitate — 14 %; lipsa personalului specializat — 35 %. **Pedagogice**: (lipsa unor obiective clare — 20 %; cercetare şi evaluare insuficientă — 46 %; instruirea personalului educaţional — 40 %; rezistenţa la schimbări — 30 %).

Utilizarea informaticii în edcaţie pe diferite stagii şi tipuri de instruiri este redată sintetic în matricea:

Tip/Stagiu	Alfabetizare %	Instruire informatică %	Mijloace învăţare %	Instruire instructori, prof. %	Management, administraţie %
Terţiar	50/22*	57/22	27/30	30/30	38/34
Secundar	38/27	22/33	0/50	24/22	19/30
Secundar tehn./specializ.	50/19	43/19	5/46	24/22	22/33
Primar	3/8	8/5	11/24	22/30	3/27
Preşcolar	3/24	—	5/46	8/5	3/8
Special	50/19	3/—	3/16	11/14	3/16
Adulţi (ed. cont.)	16/27	14/22	3/19	11/19	16/19

* Generalizat/Pilot.

Ţările membre sînt: Anglia, Argentina, Belgia, Bolivia, Brazilia, Bielorusia, Camerun, Canada, Republica Centrafricană, Ciad, Chile, Congo, Costa Rica, Cuba, Cipru, Cehoslovacia, Finlanda, Franţa, Gabon, R.D.G., Ungaria, India, Irlanda, Israel, Japonia, Iordania, Coreea de Sud, Kuweit, Luxemburg, Mauritius, Mozambic, Polonia, Samoa, Senegal, Spania, Srilanka, Elveţia, Surinam, Siria, U.R.S.S., Iugoslavia.

Cooperarea între statele membre, mijlocită şi de Intergovernamental Informatics Programme (IIP), include schimburi de informaţii şi experienţe; schimb de software, instruirea specialiştilor şi profesorilor, cercetare; asistenţă tehnică şi financiară.

Ancheta USEIT (Use in Systems of Education of Information Technologies) a UNESCO este utilă în aprecierea stadiului național și la noi*. Anchetele complementare privind **editorii de software** au reliefat tipurile de pachete de software, evaluarea și proporția lor (software proiectat pentru exerciții; programe tutoriale — învățare interactivă, — simulare; prelucrare texte, tabelărie, baze de date, CAD, sisteme expert; jocuri, altele. **Fabricanții de hardware** au dat indicații procentuale, menționate anterior.

În **„Comparative Study on Criteria and Procedures for the Evalution of Educational Software**, editat de Richard N. Tuker, prin Nederlandes Institut Voor Audio-visuele Media, prin contract între International Council for Educational Media (ICEM) și UNESCO sept. 1988, la care au participat Canada, Japonia, Ungaria, Italia, Olanda, Scoția, Anglia, S.U.A., s-au analizat **criteriile de evaluare a software-ului educațional**, definind 3 tipuri de pachete: 1) — **Programe profesionale** (inițial produse pentru comunități de afaceri sau profesionale). 2) — **Programe deschise sau de aplicații** (proiectate pentru educație — conținut liber). 3) — **Programe didactice** (conținut specific educațional).

Evaluarea apare în trei faze: a) **Testare în producția produselor**. b) **Selecția prin translație și conversie** (în țările mici, la achiziționarea și traducerea programelor). c) **Selecția de către autorități și școli, deci de către utilizatori**. Deci se poate vorbi de o evaluare **formativă** (în producție) și **sumativă** (la utilizatori).

Statele au metode întrucîtva diferite de a evalua software-ul educațional, toate bazate pe tehnici ale mulțimilor fuzzy (vagi), **adică pe chestionare cu acordarea de note pe diverse secțiuni**: (A — întrebări considerate înaintea utilizării pachetelor de programe; B — întrebări considerate în timpul utilizării programelor; C — întrebări după utilizarea programelor).

Pachetul educațional trebuie să includă documentația educațională și tehnică, iar răspunsurile trebuie să se dea la următoarele:

A. Intrebări înaintea utilizării pachetelor

Obiective și scopuri educaționale

1. Sînt clare enunțurile obiectivelor educaționale?
2. Pentru ce subiecte și pentru ce studenți se utilizează?
3. Sînt obiectivele pachetului relevante în cursuri și instituții pentru studenții utilizatori; există enunțuri asupra **abilității cerute** utilizatorilor?
4. Sînt enunțate clar enunțurile **stilului de instruire** (exerciții — repetare, simulare, interactiv-tutorial ș.a.) utilizat în program?
5. Este stilul de instruire adecvat obiectivelor cursurilor, instituției, profesorului și elevilor?
6. Pachetul este pentru cazul clasei, grupurilor, studenților individuali sau profesorului?
7. Există indicații privind modul în care poate fi apreciată atingerea obiectivelor de către pachet?
8. Dacă programele pot fi utilizate fără asistență, au fost asigurate notițe pentru studenți, pentru o instruire acceptabilă?
9. Este clar ce activitate pre-, post- și în lucrul cu calculatorul este necesară și există material suficient pentru profesor sau student pentru această activitate?

Cerințe tehnice

1. Sînt clare enunțurile privind ce hardware este necesar pentru ca programul să ruleze (calculatorul, memoria, periferiile necesare)?
2. Sistemul de operare și alte cerințe software sînt indicate?

B Intrebări în timpul utilizării programului

Proiectare educațională

1. Este dată o introducere cu numele pachetului, urmată de o secțiune opțională cu instrucțiuni de utilizare?

* A se vedea în continuare.

2. Dacă pachetul poate fi utilizat cu o largă categorie de abilităţi la diferite nivele de dificultate incluse?
3. Este utilizatorul implicat într-o cale activă sau relevantă în interacţiunea cu programul?
4. Este adoptat un stil „conversaţional" (fraze şi mesaje mai curînd scurte decît lungi)?
5. Sînt mesajele variate şi prietenoase?
6. Dacă programul poate să răspundă precis şi fiabil la diferite întrebări corecte privind o problemă particulară?
7. Sînt permise formate fără restricţie sau răspunsuri lungi?
8. Este utilizatorul pus să introducă mari volume de informaţii?
9. Dacă se utilizează formatul cu selecţii multiple pentru chestiunea pusă şi răspunsul solicitat?
10. La sfîrşitul pachetului există un sumar din care utilizatorul să deducă ce a învăţat şi ce activitate mai trebuie depusă?

Prezentare şi amplasare

1. Este prea mult text prezent pe ecran?
2. Sînt marginile şi spaţierile bine utilizate pentru înţelegere uşoară?
3. Sînt utilizate adecvat sublinierile, cartuşele, afişările intermitente şi culorile?
4. Sînt folosite efectiv sunetele şi graficele şi care sînt scopurile educaţionale clare în utilizarea lor?

Controlul şi ghidarea instruirii

1. Trebuie ca elevul să verifice succesiunea în program sau programul singur „merge" la o nouă pagină sau secţiune?
2. Trebuie ca utilizatorul să revină la pagini anterioare repetat sau cheamă utilitarul HELP? Există o cale de ieşire din program?
3. Pot elevii să corecteze uşor erorile?
4. Este întotdeauna evident pentru utilizator, în orice punct din program, care sînt opţiunile disponibile şi ce trebuie să facă?
5. Sînt opţiunile disponibile într o manieră consistentă în tot cursul programului?
6. Există o reacţie semnificativă cînd se dau răspunsuri greşite?
7. Cînd se dau răspunsuri corecte se asigură o reacţie pozitivă?

Robusteţe tehnică

1. Se poate distruge programul prin manipularea greşită a unei taste?
2. Sînt dezactivate toate tastele cu excepţia aceleia ce trebuie activată;?
3. Sînt protejaţi utilizatorii de mesajele sistemului?

C Întrebări după utilizarea programelor

1. Care sînt obiectivele atinse (A) prin utilizarea programului? Să se revadă răspunsurile din prima secţiune din A în lumina experienţei programului — au fost validate pretenţiile ridicate faţă de program?
2. Au fost atinse alte obiective valoroase (nepropuse) prin folosirea programului?
3. Pentru atingerea obiectivelor a fost eficient utilizat calculatorul? A exploatat pro gramul capabilitatea calculatorului (memoria, manipularea unei mari cantităţi de date, generarea unor grafice dinamice, interacţiunea cu calculatorul etc.) Ar fi putut fi atinse obiectivele pachetului, mai ieftin şi mai convenabil, fără utilizarea calculatorului?
4. Este programul proiectat modular, pentru a permite modificări?
5. A fost pachetul testat în domeniu cu auditoriul avut în vedere? Dacă da, care au fost rezultatele testării?
6. Au fost publicate în revistele educaţionale menţionări asupra pachetului? Dacă da, menţionaţi.
7. Cel care a dezvoltat sau publicat programul trebuie să fie rugat a da numele utilizatorilor curenţi ai pachetului?
8. Este clar dacă sînt necesare activităţi viitoare? Dacă da, este furnizat suficient material studentului sau profesorului, pentru a desfăşura această activitate?

● Calculatorul HC - 85 extins, cu interfață de disc flexibil, interfață serială (conectare imprimantă etc.) și interfață de rețea ● Calculatorul HC-88 cu dublă compatibilitate: HC-85 extins și CP/M ● Operarea cu discul flexibil; operarea cu periferice; operarea în rețele locale; joc de rețea ● BASIC 85 extins ● Manipularea fișierelor ● Apelurile funcțiilor interfețelor din limbaj de asamblare.

Calculatorul personal HC-85 extins.

Microcalculatorul HC-85 în configurație extinsă este echipat cu o plachetă suplimentară, cuplată prin conectorul de extensie cu placa de bază. Această plachetă inglobează trei interfețe: pentru unitatea singulară sau duală de disc flexibil, pentru o linie serială standard RS-232-/CCITT V24 și pentru cuplarea microcalculatoarelor HC într-o rețea, folosind o pereche de fire torsadate.

Cuplarea unor unități duale de discuri flexibile de 5 1/4" asigură o capacitate externă de memorare de 1,28 Mo, pentru maximum 128 fișiere distincte.

Interfața serială permite cuplarea la HC a unei imprimante sau interconectarea cu un alt calculator.

Interfața pentru cuplarea în rețea oferă posibilitatea legării pînă la 64 de sisteme, oferind astfel o soluție pentru aplicațiile in domeniul invățămintului, în laboratoarele școlare de informatică.

Toate aceste noi facilități hardware de care dispune HC-85 in configurație extinsă posedă un suport software, prin extensia instrucțiunilor limbajului BASIC - HC.

Noile instrucțiuni, cît și modurile de operare cu noile facilități sint prezentate intr-o anexă.

Calculatorul personal HC-88.

Microcalculatorul HC-88, cu dubla compatibilitate, reprezintă o

soluție care inglobeaza atît caracteristicile sistemului HC-85 cît și pe cele ale microsistemului CUB-Z, ca mașină CP/M. HC-88 s-a conturat ca o soluție de inlocuire in producția Intreprinderii de Calculatoare Electronice București a sistemelor CUB-Z.

In configurația maximă, HC-88 constă din urmatoarele echipamente:

- HC de baza format din:
  - echipamentul nucleu (µP Z80A; 80 K RAM(64+16), 2K ROM)
    . placa de bază,
    . sursa de alimentare,
    . monitorul monocrom,
- tastatura,
- doua unități de disc flexibil de 5 1/4",
- imprimantă matricială grafică,
- monitor color,
- codor RGB/PAL.

Monitorul monocrom si monitorul color pot fi cuplate simultan la calculator.

Optional, echipamentul nucleu HC-88 poate conține:

- placheta IO/EPROM,
- placheta IO/SIO,
- modul extensii EUROCARD,
- placheta programator EPROM
- placheta extensie memorie RAM 256 Ko,
- placheta extensie memorie RAM 1 Mo.

Conectorul serial permite legarea unui cablu de adaptare pentru casetofon.

CONTINUARE IN VOLUMUL 2
LA PROLOG—DIALOG—EPILOG

Cîte ceva despre carte:

O prezentare sistematică există în **studiul introductiv**, o descriere succintă — pe **coperţi**, structura detailată — în **tablele de materii** din volumele 1 şi 2. Remarci privind **caracteristicile ce o fac utilă informatizării învăţămîntului** nostru se dau în paginile XV—XVI din vol. 1, unde se menţionează şi **întîrzierea apariţiei** sale. Evidenţiem trei factori care au concurat la începerea elaborării sale în 1986: un ciclu al Editurii Tehnice, cu apariţii anterioare, dedicat de redacţie calculatoarelor personale; propunerea globală primită de la cercurile de informatică extraşcolare pentru elevii din clasele I—VIII; dorinţa de a scrie a unor autori anteriori — creatori şi utilizatori de calculatoare personale. În 1987/88, a fost gata pentru tipografie o variantă a lucrării, cu un titlu născocit de redacţie (abc de calcul electronic), pentru a o feri de hotărîrea aberantă de la începutul anului 1988 a „academicienei-analfabete-dictatoare", care elimina de la apariţie toate cărţile despre calculatoare, cu o interdicţie expresă privind **calculatoarele personale**, ce îi apăreau drept cele mai „nocive" cuvinte pentru popor.

Cîte ceva despre autori:

ADRIAN-CRISTIAN PETRESCU, profesor la Catedra de calculatoare din Institutul Politehnic Bucureşti; absolvent al facultăţii de Electronică şi Tc, iar din 1964 doctor inginer în domeniul calculatoarelor. Din 1971, Fellow of British Computer Society. A condus şi a participat efectiv la proiectarea şi realizarea mai multor sisteme de calcul: MAC I şi MAC II — (Premiul MEI — 1965) FELIX MC 8 (Premiul Traian Vuia al Academiei R. S. România — 1975). FELIX M18, FELIX M118, FELIX M-216, aMIC, HC-85, FELIX-PC (Premiul I la Concursul de Creaţie Ştiinţifică şi Tehnică din anul 1987 pentru Felix PC şi premiul II pentru HC-85). Este autor al lucrărilor: **Calculatoare automate şi programare** (Ed. I — 1970 Ed. a II-a — 1974) — Editura Didactică şi Pedagogică, **Microprogramare — Principii şi aplicaţii**, Editura Tehnică, 1975; coautor şi coordonator al lucrărilor, **Microcalculatoarele FELIX M18, M18B, M118** — Editura Tehnică 1984, **Totul despre aMIC** — Editura Tehnică 1985 şi coautor — editor la lucrarea „**Calculatoarele electronice din generaţia a cincea**", în Editura Academiei R. S. România, 1985.

NICOLAE ŢĂPUŞ, şeful catedrei de calculatoare, Facultatea Automatică din Institutul Politehnic Bucureşti. Absolvent al Facultăţii de Automatică (1972), iar din 1982 doctor inginer în domeniul calculatoarelor. A participat la proiectarea şi implementarea sistemelor de calcul româneşti, FELIX MC8 (premiul Traian Vuia al Academiei R.S.R., 1975), FELIX M18, FELIX M118, FELIX M216, FELIX PC (premiul I la concursul de Creaţie Ştiinţifică şi Tehnică din anul 1987). Este autor şi coautor al unor lucrări de specialitate (manuale, articole) destinate studenţilor şi utilizatorilor de sisteme de calcul. Este coautor al lucrării **Microcalculatoarele FELIX M18, M18B, M118**, Editura Tehnică, 1984.

TRANDAFIR MOISA, şef de lucrări la catedra de calculatoare, Facultatea Automatică din Institutul Politehnic Bucureşti. Absolvent al Facultăţii de Automatică (1973), iar din 1982 doctor inginer în domeniul calculatoarelor. A participat la proiectarea şi implementarea sistemelor româneşti de calcul: FELIX MC8, (premiul Traian Vuia al Academiei R.S.R— 1975, FELIX M18, FELIX M118, FELIX M216, FELIX PC (premiul la concursul de Creaţie Ştiinţifică şi Tehnică din anul 1987). Este autor şi coautor al unor lucrări de specialitate (articole, manuale) destinate specialiştilor din domeniul tehnicii de calcul. Este coautor al lucrării **Microcalculatoarele FELIX M18, M18B, M118**, Editnra Tehnică, 1984.

GHEORGHE N. RIZESCU n. 1928 Călina, Vîlcea, absolvent al Facultăţii de Matematică şi Fizică din Bucureşti în 1952, profesor emerit. În 1988 iese la pensie ca **director** al Liceului „Dimitrie Cantemir" din Bucureşti. Ca profesor şi ca **redactor** la Gazeta Matematică este autor de probleme, articole şi comunicări cu caracter ştiinţific sau metodico-ştiinţific; autor al lucrării „**Laboratorul de matematică — teme şi fişe experimentale**, O.I.D.—I.C.P.E., Bucureşti, 1978. De asemenea, este coautor la o serie de lucrări publicate în revista „Studii şi Cercetări Matematice", Editura Academiei R.S.R., la lucrarea „**Laboratorul de Matematică**, E.D.P., 1973 şi la lucrările „**Teme pentru cercurile de matematică din licee**", vol. I (1977, E.D.P.) şi vol. II (1980, E.D.P.). Coautor la manualul de matematică „**Algebra pentru clasa a IX-a**", E.D.P. — 1978, ... şi la volumele I şi II „**Totul despre calculatorul personal aMIC**", Editura Tehnică, 1985. Autor a numeroase articole şi comunicări privind informatica şi învăţămîntul. S-a preocupat îndelung, cu rezultate notabile, de modernizarea tehnologiei învăţămîntului.

VIORICA ELENA HĂRĂBOR, analist la Institutul de cercetări pentru Informatică (ICI), Bucureşti. Absolventă a Facultăţii de Matematică a Universităţii din Bucureşti (1970). Se ocupă de utilizarea calculatoarelor personale şi de iniţierea copiilor în informatică. A publi-

Datorită complexității prevăzute pentru ansamblul cărți-casete magnetice, a unui colectiv ce a atins 8 persoane din 4 unități diferite de învățămînt—cercetare—proiectare—producție, a unei structurări ambițioase, **procesul de editare a fost lung**, cu numeroase iterații de îmbunătățire și corelare a elaborărilor parțiale, de selectare a sutelor de module și pachete de programe cum și de includere a unor lucrări valoroase de programare ale unor elevi și studenți. Mari întîrzieri au fost provocate de lipsa de hîrtie și de bandă magnetică pentru casete; după multă vreme consumată și multe stăruințe depuse de redacție și de unii dintre autori, s-au obținut materialele și în dec. 1989, am ajuns la tipărirea unei mari părți a lucrării. După **revoluția din decembrie 1989** (în condiții la care ne referim în pag. VIII—IX din vol. 2) cu un colectiv lărgit — de 11 persoane, folosind documentație străină, nepermisă și necunoscută în 1989, noile structuri ale învățămîntului, cercetării, noile realizări ale industriei de c.p. — interzise și ele anterior — am eliminat și am reelaborat, retipărit și tipărit peste 200 pagini, care perfecționează substanțial lucrarea și pregătește primul tiraj pentru întîlnirea cu publicul în mai 1990, sub titlul său real.

---

cat articole (inclusiv în seria AMC a Editurii Tehnice) și a prezentat comunicări științifice pe această temă, în țară și în străinătate.

MIHAI MĂRȘANU, șef al colectivului de cercetare pentru sisteme cu microcalculatoare din Institutul de Tehnică de Calcul, București. Absolvent al Facultății de Electronică și Telecomunicații din Institutul Politehnic București în 1967. Doctorand. Este autor și coautor al multor lucrări și comunicări științifice destinate specialiștilor din domeniul calculatoarelor, inclusiv **„Calculatoarele electronice din generația a cincea"**, Editura Academiei, 1985. A participat în 1986 și 1987 la o serie de acțiuni de inițiere a elevilor în domeniul calculatoarelor personale. Preocupări: sistemele multimicro, mașinile de baze de date și optimizarea accesului la date stocate în memorii rotaționale.

IOAN PAUL ZAMFIRESCU, absolvent al Facultății de Electrotehnică din Institutul Politehnic București, doctorand în calculatoare-automatizări. Activitate îndelungată ca editor de cărți și reviste la Editura Tehnică, în domeniile automatică–informatică–electronică–management. Preocupări în analiza și proiectarea asistată de calculator a sistemelor de editare–tipărire–difuzare. Specializare în informatică în Franța și proiecte pentru Hachette — Paris și I.B.M. — France. Autor de articole și de cărți.

EUGEN DOBROVIE, absolvent al Facultății de Automatică din Institutul Politehnic București în 1977. Lucrează la Întreprinderea de Calculatoare Electronice; a făcut parte din colectivele de proiectare ale calculatoarelor M118, HC–85, HC–88; a fost instructor specialist în taberele de informatică pentru elevi.

VICTOR CRISTIAN COSOSCHI, absolvent al Facultății de Automatică, Secția Calculatoare, din Institutul Politehnic București, în 1977. Lucrează de atunci la Întreprinderea de Calculatoare Electronice, contribuind în colectivele de proiectare hardware și software la realizarea sistemelor Felix M18, CUB–01, VDT 40C, HC–85, HC–88.

NICOLAE BADEA, cercetător științific la ICI București. Absolvent (1971) al Facultății de Cibernetică Economică din București; doctorand în domeniul informaticii. Autor de comunicări la manifestări științifice interne și internaționale. A contribuit la proiectarea și implementarea de sisteme și programe informatice pentru fabricație asistată de calculator, proiectare asistată, utilizarea calculatoarelor în conducere, educație. Este coautor la lucrările: **Noile tehnologii de vîrf și societatea** (1983). **Informatizarea societății — un fenomen inevitabil?** Editura Științifică și Enciclopedică 1985, **Inginerie de sistem, automatică și informatică în transporturi** vol. 1 și vol. 2, Editura Tehnică, 1988—1989.

TRAIAN MIHU, director tehnic al Întreprinderii de Calculatoare Electronice, București; absolvent al Institutului Politehnic București, Facultatea Automatică, secția calculatoare (1970). În cadrul Centralei Industriale pentru Electronică și Tehnică de Calcul a coordonat programele de asimilare pentru majoritatea sistemelor de calcul, terminalelor și echipamentelor periferice care sînt în producția actuală.

Prezent cu comunicări științifice, privind echipamente și tehnici de înregistrare magnetică, în cadrul unor sesiuni interne și internaționale.

CONSTANȚIN HĂRĂBOR, absolvent al Facultății de Matematică a Universității din București, în 1970, în prezent profesor la liceul Dimitrie Bolintineanu, București. Preocupări în utilizarea calculatoarelor personale în cercuri, tabere și în concursuri de matematică și informatică. Coautor la culegeri de probleme. Comunicări științifice în țară și în străinătate.

CUPRINS

Volumul 1

— *Studiu introductiv* (Academician Prof. Nicolae Teodorescu) V

— Cuprins general (vol. 1 şi 2) XI

— PROLOG—DIALOG—EPILOG (continuă în vol. 2) XIII

— Cuprins detailat vol. 1 XXVIII

Partea I: **Calculatoare, microcalculatoare şi calculatoare personale în ţara noastră şi pe plan mondial** 1

Capitolul 1. **Tehnica de calcul şi informatica în România. Începuturi, evoluţie, aplicaţii** 1

1.1. Contribuţii şi priorităţi româneşti în matematică, cibernetică, electronică, tehnică de calcul şi informatică 2
1.2. Evoluţia generaţiilor de calculatoare în ţara noastră şi în lume 5
1.3. Industria românească de calculatoare electronice. Informatica .. 12
1.4. Aplicaţii ale calculatoarelor în societatea modernă 17

Capitolul 2. **Caracterizarea şi evoluţia calculatoarelor personale şi microcalculatoarelor** 21

2.1. Ce este un calculator personal? .. 21
2.2. Microelectronica pe plan mondial 24
2.3. Calculatoare personale, personal-profesionale şi microcalculatoare în lume 28
2.4. Industria românească de calculatoare personale şi personal-profesionale 35

Partea a II-a: **Calculatoare numerice. Realizare fizică — bazele aritmetice şi logice** 37

Capitolul 3. **Baze aritmetice** 37

3.1. Sistemul de numeraţie binar .. 37
3.2. Conversia binar-zecimală 39
3.3. Conversia zecimal-binară 39
3.4. Reprezentările octală şi hexazecimală 40
3.5. Codul binar-zecimal (2—10) .. 41
3.6. Adunarea şi înmulţirea numerelor binare 42
3.7. Reprezentarea numerelor negative 43
3.7.1. Reprezentarea în modul şi semn 43
3.7.2. Reprezentarea în complementul faţă de unu .. 43
3.7.3. Reprezentarea în complementul faţă de doi .. 44
3.8. Reprezentarea numerelor reale 45
3.8.1. Reprezentarea în virgulă fixă 45
3.8.2. Reprezentarea în virgulă mobilă 46
3.9. Reprezentarea caracterelor alfanumerice 47

Capitolul 4. **Logica matematică şi circuitele logice** 49

4.1. Generalităţi 49
4.2. Formele canonice ale funcţiilor logice 51
4.3. Identităţi şi simplificări 54
4.4. Realizarea fizică a funcţiilor logice 54
4.4.1. Circuite logice combinaţionale 54
4.4.2. Circuite logice secvenţiale 60

Partea a III-a: **Calculatorul personal HC-85. Structură, componente, operare, programare** 66

Capitolul 5. **Structura şi modul de operare ale calculatoarelor numerice** 66

5.1. Conceptul de calculator 66
5.2. Structura şi operarea calculatoarelor numerice 68
5.3. Instrucţiunile calculatoarelor numerice 74
5.4. Software, programare, algoritmi 77

Capitolul 6. **Structura şi componentele microcalculatorului HC-85** 80

6.1. Configuraţie 80
6.2. Unitatea centrală: microprocesorul Z 80, memoriile RAM, ROM 81
6.3. Subsistemul de intrări-ieşiri: interfeţe cu televizorul, tastatura, casetofonul, difuzorul; conectorul de extensii 87

Capitolul 7. **Elemente de programare în limbaj algoritmic** 93

7.1. Algoritmi 93
7.2. Limbajul algoritmic (pseudocod) 94

Partea a IV-a: **Programarea în limbajul BASIC pe calculatorul HC-85** ... 103

Capitolul 8. **Caracteristicile şi elementele limbajului BASIC** 103

8.1. Noţiuni introductive 103
8.2. Tastatura 105
8.3. Moduri de lucru 106
8.3.1. Modul de lucru K 106
8.3.2. Modul de lucru L 106
8.3.3. Modul de lucru C 108
8.3.4. Modul de lucru E 108
8.3.5. Modul de lucru G 109
8.4. Alfabetul limbajului BASIC .. 111
8.4.1. Setul de caractere 112
a Litere 112
b Cifre 112
c Operatori aritmetici .. 112
d Operatori de relaţie .. 112
e Operatori logici 112
f Semne de punctuaţie .. 112
g Semne speciale 113
8.4.2. Constante 114
1°Constante numerice .. 114
2°Constante alfanumerice 116
8.4.3. Variabile 116
1°Variabile numerice .. 117
2°Variabile alfanumerice 118
8.4.4. Expresii 119
1°Expresii numerice 119
2°Expresii alfanumerice 120
8.4.5. Funcţii standard 120
1°Funcţii standard matematice 120
2°Funcţii standard pe şiruri de caractere 121

Capitolul 9. **Instrucţiunile limbajului BASIC (tratare detaliată cu exemplificări)**.... 122

9.1. PRINT 123
9.2. RUN 129
9.3. DELETE 130
9.4. GOTO 131
9.5. CLS 131
9.6. NEW 132
9.7. LIST 132
9.8. PRINT AT 133
9.9. PRINT TAB 134
9.10. REM 135

9.11. LET 135
9.12. BEEP 137
9.13. PAUSE 139
9.14. LOAD 140
9.15. SAVE 141
9.16. VERIFY 142
9.17. MERGE 143
9.18. INPUT 143
9.19. INPUT LINE 145
9.20. PLOT 146
9.21. DRAW 147
9.22. FOR-NEXT 152
9.23. RND 154
9.24. RANDOMIZE 154
9.25. CIRCLE 155
9.26. OVER 156
9.27. IF 158
9.28. STOP 160
9.29. CONTINUE 161
9.30. DIM 162
9.31. DATA 161
9.32. READ 164
9.33. RESTORE 165
9.34. DEF FN 166
9.35. GOSUB-RETURN 167
9.36. PEEK 169
9.37. POKE 169
9.38. FLASH 170
9.39. PAPER 170
9.40. INK 171
9.41. INVERSE 172
9.42. BORDER 172
9.43. POINT 173
9.44. ATTR 173
9.45. LPRINT 174
9.46. LLIST 174
9.47. COPY 174

Partea a V-a: **Programarea în limbajul LOGO pe calculatorul HC-85** 175

Capitolul 10. **Caracteristicile și elementele limbajului LOGO** 175

10.1. Evoluția limbajelor de programare 176
10.2. Ce este LOGO? Caracteristici .. 178
10.3. LOGO în școli. De ce și cum? .. 182
10.4. LOGO pe HC-85 184
10.5. Reguli gramaticale ale limbajului LOGO 186
10.6. Obiectele limbajului LOGO .. 187
Numere 187
Cuvinte 188
Liste 189
Delimitatori 190
Variabile și atribuirea numerelor în LOGO 191
Variabile locale și globale 193
10.7. Reguli privind lucrul cu proceduri 194
Proceduri cu intrări 195
Tipuri de proceduri 196
Introducerea și editarea procedurilor 197
10.8. Expresii condiționale 198
Instrucțiunea IF și predicate .. 198
Predicate predefinite 199
Predicate definite de utilizator 199
10.9. Linii complexe și interpretarea lor 200
10.10. Glosar de termeni LOGO 202

Capitolul 11. **Primitivele limbajului LOGO, cu exemplificări** 207

11.1. Caracteristici generale ale primitivelor 207
11.2. Primitive LOGO pentru controlul penelului și ecranului în regim grafic 208
11.2.1. Primitive pentru schimbarea stării penelului
BACK 209
CLEARSCREEN 210
FORWARD 210
HIDETURTLE 211
HOME 211
LEFT 211
RIGHT 212
SETHEADING 212
SETPOS 213
SETX 213
SETY 213
SHOWTURTLE 214
11.2.2. Primitive care specifică starea penelului 214
HEADING 214
POSITION 215
SHOWNP 215
TOWARDS 215
XCOR 216
YCOR 216
11.2.3. Primitive pentru utilizarea peniței electronice și a ecranului 216
CLEAN 216
DOT 217
FENCE 217
PENDOWN 218
PENERASE 218
PENREVERSE 218
PENUP 219
SETBG 219

SETBORDER 219
SETPC 220
SETSCHRUNCH 220
WINDOW 221
WRAP 222
11.2.4. Primitive care specifică starea peniţei şi a ecranului 222
BACKGROUND 222
PENCOLOUR 223
SCRUNCH 223
11.3. Primitive LOGO pentru controlul ecranului în regim alfanumeric 223
BRIGHT 224
CLEAR TEXT............ 224
COPYSCREEN 225
CURSOR 225
FLASH 225
INVERSE 225
NORMAL 226
OVER 226
SETCURSOR 227
SETTC 227
TEXTSCREEN 227
11.4. Primitive LOGO care specifică operaţii matematice şi logice .. 228
11.4.1. Operatori aritmetici şi logici + – * / < > = ... 228
11.4.2. Primitive pentru operaţii matematice 231
ARCCOS 231
ARCCOT 231
ARCSIN 232
ARCTAN 232
COSINE 232
SINE 232
COTANGENT 232
DIV 232
INT 233
PRODUCT 233
RANDOM 233
REMAINDER 234
ROUND 234
SQRT 234
SUM 235
TANGENT 235
11.4.3. Primitive pentru operaţii logice 235
AND 236
NOT 236
OR 236
11.5. Primitive LOGO pentru lucrul cu cuvinte şi liste 236
11.5.1. Primitive care specifică preluarea unor elemente din cuvinte sau liste 238
BUTFIRST 238
BUTLAST 238
FIRST 239
LAST 239
ITEM 239
11.5.2. Primitive care concatenează cuvinte şi liste .. 240
FPUT 240
LIST 240
LPUT 241
SENTENCE 241
WORD 242
11.5.3. Primitive care examinează cuvinte şi liste .. 243
ASCII 243
CHAR 244
COUNT 244
EMPTYP 244
EQUALP 244
LISTP 245
MEMBERP 245
NUMBERP 246
WORDP 246
11.6. Primitive LOGO care lucrează asupra variabilelor 246
MAKE 247
NAMEP 248
THING 248
11.7. Primitive LOGO pentru comunicaţia cu exteriorul (echipamentele de intrare / ieşire) 248
11.7.1. Primitive care citesc informaţii din exterior .. 249
KEYP 249
READCHAR 249
READLIST 249
11.7.2. Primitive care afişează pe ecranul calculatorului 250
PRINT 250
SHOW 250
TYPE 251
11.7.3. Primitive pentru generare de sunete 251
SOUND 251
11.8. Primitive LOGO care asigură ramificaţia în program 252
11.8.1. Primitive pentru ramificaţie condiţionată .. 252
IF 253
11.8.2. Primitive pentru întreruperea procedurilor ... 253
BYE 253
OUTPUT 254
STOP 255
WAIT 256
TOPLEVEL 256
11.8.3. Primitive pentru execuţia şi repetarea unei liste de instrucţiuni .. 257
REPEAT 258
RUN 258

11.9. Primitive pentru lucrul cu fişiere 259
11.9.1. Primitive pentru încărcarea de pe casetă 260
LOAD 260
LOADD 260
LOADSCR 260
11.9.2. Primitive pentru salvarea pe casetă 260
SAVE 261
SAVEALL 261
SAVED 261
SAVESCR 261
11.9.3. Primitive pentru controlul imprimantei şi tipărirea informaţiei de pe ecran 261
PRINTON 262
PRINTOFF 262
COPYSCREEN 262
11.9.4. Primitive pentru lucrul cu microdriverul 262
SETDRIVE 263
CATALOG 263
ERASEFILE 263
11.10. Primitive pentru gestiunea spaţiului de lucru ... 263
11.10.1. Primitive pentru examinarea spaţiului de lucru 264
NODES 264
RECYCLE 264
11.10.2. Primitive pentru afişarea conţinutului spaţiului de lucru .. 264
PO 265
POALL 265
PONS 265
POPS 265
POTS 265
11.10.3. Primitive pentru ştergerea informaţiei din spaţiul de lucru 266
ERALL 266
ERASE 266
ERN 266
ERNS 267
ERPS 267
11.11. Primitive pentru definirea şi modificarea procedurilor 267
11.11.1. Primitive pentru definirea şi editarea procedurilor 267
EDIT 267
EDNS 269
TO 269
END 270
11.11.2. Primitive pentru modificarea procedurilor sub controlul programului 270
COPYDEF 270
DEFINE 270
DEFINEP 271
PRIMITIVEP 271
TEXT 271
11.12. Primitive pentru funcţii diverse 272
11.12.1. Primitive pentru accesul la memoria calculatorului 272
● BLOAD 272
● BSAVE 272
● CALL 273
● DEPOSIT 273
● EXAMINE 274
● RESERVE 274
● RESERVED 275
11.12.2. Primitive pentru lucrul cu interfaţa serială 275
● SETSERIAL 275
● SERIALIN 276
● SERIALOUT 276
11.12.3. Primitive pentru specificarea obiectelor şi primitivelor LOGO 276
● CONTENTS 276
● PRIMITIVES 276

Capitolul 12. Tehnici de programare în LOGO şi aplicaţii 277

12.1. Proiectarea programelor 277
12.2. Recursivitatea în LOGO 284
12.3. Mici aplicaţii în LOGO 298
12.3.1. Exemple grafice 298
12.3.2. Suma a două numere aleatoare 301
12.3.3. Ordonare alfabetică .. 302
12.3.4. Conversia numerelor naturale din baza 10 într-o bază mai mică decît 16 303
12.3.5. Conversia dintr-o bază oarecare în altă bază .. 304
12.3.6. Desenează interactiv .. 307
12.3.7. Turnurile din Hanoi 309
12.4. Mesaje de eroare în LOGO 312

Partea I

CALCULATOARE, MICROCALCULATOARE ŞI CALCULATOARE PERSONALE ÎN ŢARA NOASTRĂ ŞI PE PLAN MONDIAL*

Capitolul 1 | Tehnica de calcul şi informatica în România. Începuturi, evoluţie, aplicaţii.

Încă din anii '60, dar şi la începutul anilor '70, ilustrul profesor şi renumitul matematician Grigore C. Moisil a publicat o serie de articole prin care leagă matematica de ştiinţa informaticii, dar şi de cultura generală, pledînd pentru informatică şi învăţarea ei. Dintr-unul din aceste articole din 1971, care constituie prefaţa volumului 13—14 al seriei continue AMC (Automatică, Metrologie, Calculatoare) a Editurii Tehnice, am selectat cîteva idei sugestive pentru cei tineri şi nu numai pentru ei:

> Problema calculatoarelor nu este numai problema construcţiei lor; ea este în primul rînd problema utilizării lor.
>
> Un calculator costă atît de mult, încît orice pierdere de timp din cauză că programarea este neîndemînatec făcută trebuie să fie evitată. Cei ce lucrează la calculator trebuie să fie deosebit de bine instruiţi. O cultură matematică insuficientă costă mult. Un calculator utilizat de un om nu îndestul de priceput nu se strică, e mai rău: lucrează cu randament scăzut. E mai rău zic, fiindcă o maşină care stă, se vede că stă; toată lumea o vede că stă. O uzină trebuie să dea o anume producţie; dacă nu o dă, se vede că uzina e prost utilizată. Un calculator care din cauza proastei programări dă numai a zecea parte din ce poate da, merge; nimeni nu-l vede că merge încet: fiindcă el merge la fel de repede, dar treaba pe care o face, e pus s-o facă încet... Pe bună dreptate se spune că construcţia unui calculator pune în evidenţă toată capacitatea tehnică a celor ce îl construiesc, nu numai capacitatea tehnică în construcţia calculatoarelor.

---

* Atragem atenţia cititorilor că pentru a ilustra familiile de calculatoare personale şi personal-profesionale, ca şi aplicaţiile lor în învăţămînt şi în alte domenii, cartea cuprinde un grupaj de pagini cu felurite anexe, explicate, care au legătură cu textul din partea I (ca şi cu textele din alte părţi ale cărţii). A se vedea şi secţiunea PROLOG-DIALOG-EPILOG.

Dar tot atît de mult utilizarea unui calculator pune în evidență toată capacitatea științifică a celor ce îl întrebuințează. Cunoștințele de logică formală sînt tot atît de esențiale pentru un informatician ca și cele de analiză numerică...

Din planul de învățămînt al specializării în „Știința Calculatoarelor" dat în acest număr se vede că cultura de bază a celui ce trebuie să se ocupe de „software" e aceea a unui matematician instruit în matematica modernă: abstractă, algebrică, formală.

Cei ce organizăm învățămîntul în acest domeniu, cei care predăm cursuri asupra acestor discipline vom medita mult asupra paginilor ce urmează.

...Nu putem crede că această muncă se poate duce fără a ști ce fac ceilalți. Editura tehnică a înțeles acest lucru și numărul cărților traduse și publicate de această editură e mare. După cum noi nu trebuie să credem că un om sau un popor poate progresa lucrînd izolat, fără să știe ce fac ceilalți, tot astfel nu trebuie să ne închipuim că putem cu forțele noastre numai, să acoperim toată producția de cărți pe care e nevoie să le citim. Un număr mare de traduceri de cărți bune, din toate limbile, contribuie la ridicarea nivelului științific. În acest caz și numai în acest fel vom putea și noi să scriem cărți bune, care să fie și ele traduse în alte limbi. În unele domenii această treaptă a fost atinsă. În domeniul Științei Calculului ea trebuie să fie atinsă, dar sîntem foarte la început.

Dar progresul științific și tehnic nu se poate face numai cu cărțile ce le putem edita. Nici un popor nu poate edita toate cărțile de care are nevoie, pe care le citește. O carte tradusă e întotdeauna în urmă cu mulți ani. Dacă cititorul nu-și dă seama de aceasta, atunci el dovedește că e cu mai mulți ani în urma cărții...

Această confruntare a muncii științifice naționale pe plan internațional e absolut necesară.

... Știința calculului nu se dezvoltă însă numai spre beneficiul ingineriei, statisticii și economiei. Lingvistica, medicina, biologia, psihologia, sociologia, arheologia, știința literaturii, toate aceste discipline umanistice cer acces la calculator. Aceasta e marea mirare a vremii noastre. Trebuie deci să pregătim învățămîntul despre calculatoare și pentru studenții facultăților științelor vieții, ale omului și ale societății...

Am fost deosebit de bucuros scriind și citind o parte a acestui număr al revistei. Cititorul, mai imparțial decît mine, va compara ce s-a făcut cu ceea ce va trebui să se facă. Va constata că ne stă în față o muncă imensă, ceea ce e înviorător.

Să ne oprim puțin asupra cîtorva din contribuțiile aduse de profesori și specialiști români de prestigiu în domenii ca: matematica, cibernetica, neurocibernetica, electronica, microelectronica și desigur în cele ale științei calculatoarelor și informaticii.

1.1. Contribuții și priorități românești în matematică, cibernetică, electronică, tehnică de calcul și informatică

MATEMATICA. Întemeietorii școlii matematice moderne românești sînt considerați savantul multilateral *Spiru Haret* (1851—1912), primul român care obține în 1878 titlul de doctor în matematică și *David Emmanuel* (1854—1941), profesor strălucit și al doilea român doctor în matematică.

Dintre iluștrii reprezentanți ai matematicii în România la începutul acestui secol, amintim pe *Traian Lalescu* (1882—1929), *Dimitrie Pompeiu* (1873—1954) și *Gheorghe Țițeica* (1873—1939), doctori în matematică, pedagogi și creatori de renume mondial.

Contibuții importante în dezvoltarea matematicii sînt datorate savanților de renume internațional *Miron Nicolescu* (1903—1975), *Simion Stoilow* (1887—1961), *Alexandru Ghica* (1902—1964) și *Alexandru Myller* (1879—1965).

Trebuie menționată contribuția originală a academicienilor *Octav Onicescu* (1892—1983) și *Gheorghe Mihoc* (1906—1981) la dezvoltarea teoriei probabilităților și statisticii matematice, începînd cu conceptul de lanț probabilistic cu legături complete, pe care l-au introdus în anul 1935. Academicianului *Octav Onicescu* îi datorăm, printre altele, primul curs din România de calculul probabilităților (1925), introducerea conceptului de energie informațională (1966), mecanica invariantivă (1954—1964) și contribuțiile la teoria jocurilor (1960).

Meritul de a forma și orienta școala românească de logică matematică cu o largă recunoaștere internațională îi revine academicianului profesor doctor docent *Grigore C. Moisil*

(1906—1973). Prima sa lucrare de logică matematică a fost publicată în ţara noastră încă din anul 1936, dar lucrarea sa fundamentală rămîne *„Teoria algebrică a mecanismelor automate"* publicată în 1959 şi reeditată în mai multe ţări. Din colectivul de cercetători în matematică pe care l-a format, s-a desprins în 1955 grupul de cercetare în domeniul logicii matematice aplicate în tehnică (teoria algebrică a automatelor) şi, ceva mai tîrziu, un grup de cercetare în domeniul logicii matematice cu aplicaţii în rezolvarea unor probleme economice (programarea pseudo-booleană). În anul 1957 iniţiază primele cursuri de calculatoare electronice, la Facultatea de Matematică din Bucureşti.

Matematicieni români contemporani au dezvoltat cercetări noi în domeniile variate ale matematicii, în informatică, mai ales în teoria algoritmilor şi limbajelor şi există preocupări intense în domeniile inteligenţei artificiale, lingvisticii matematice, teoriei jocurilor, modelării învăţării, modelării matematico-semiotice, simulării, statisticii şi teoriei probabilităţilor.

CIBERNETICA. Doctorul român *Ştefan Odobleja* (1902—1978) este un adevărat precursor al ciberneticii, primul care a evidenţiat principiile ciberneticii generalizate în lucrarea sa în două volume „Psihologia consonantistă", publicată în limba franceză în 1938—1939 şi difuzată în rîndul specialiştilor, inclusiv în străinătate.

El generalizează fenomenul circularităţii şi stabileşte printre altele, legea reversibilităţii în ştiinţă, psihologie, economie şi societate (cercul vicios de acţiuni şi reacţiuni reciproce, concept similar cu ceea ce numim azi bucla de reacţie cibernetică). Odobleja preconiza gîndirea artificială şi maşinile cu gîndire.

Alţi doi români pot fi consideraţi precursori ai ciberneticii. Academicianul medic *Daniel Danielopolu* (1884—1955) a folosit în 1928 conceptul de mecanism circular, explicînd fenomenele de reglare în organismul uman şi a evidenţiat procesele biologice informaţionale, devenind un ilustru precursor român al biociberneticii. În mod independent, prof. dr. ing. Paul Postelnicu a generalizat ulterior procesele electronice cu reacţie (conexiunea inversă sau complexul vicios, după denumirea dată iniţial) privind naşterea şi evoluţia vieţuitoarelor. Lucrările profesorului *Paul Postelnicu* au fost elaborate în anul 1944, dar trimise pentru publicare, nu au fost acceptate, astfel încît ele au apărut abia în 1968 şi 1979.

Dar, pe plan mondial, *Norbert Wiener*, rămîne acreditat cu introducerea termenului de cibernetică şi cu fundamentarea principiilor de bază, deşi a făcut aceasta mult mai tîrziu, în anul 1948, cînd a publicat în S.U.A. cartea *„Cybernetics or Control and Communication in the Animal and the Machine"*.

Astfel, România apare în mod evident, ca una din ţările care a contribuit la fundamentarea ciberneticii.

O serie de profesori români contemporani aduc contribuţii importante la dezvoltarea ciberneticii tehnice, socio-economice, la aprofundarea corelaţiilor dintre informatică, cibernetică, revoluţia tehnico-ştiinţifică şi informatizarea societăţii.

Cercetări în domeniul neurociberneticii au efectuat academician *Constantin I. Parhon* (1874—1969), profesor doctor inginer *Gheorghe Cartianu* (1907—1982) şi alţi distinşi profesori, ingineri, medici şi matematicieni contemporani.

ELECTRONICA. În perioada 1900—1960 o serie de academicieni, profesori şi inventatori din ţara noastră au pus bazele şcolii ştiinţifice româneşti în domeniul electronicii. Astfel, s-au remarcat, prin contribuţii originale în lucrările lor ştiinţifice, tehnice şi didactice:

— *Dragomir Hurmuzescu* (1865—1954) în electricitate, ca inventator al electroscopului în 1894 şi în radiocomunicaţii cu primul radioreceptor şi primele radioemiţătoare din ţara noastră, construite în 1925, respectiv 1927 şi 1928 la Institutul Electrotehnic de pe lîngă Universitatea Bucureşti.

— *Mihai Konteschweller* (1897—1947) în radiofonie şi mai ales în telemecanică, domeniu în care a realizat un prim experiment în 1934, publicînd apoi primul tratat de telemecanică în 1937.

— *Ion Constantinescu* (1884—1963) în telecomunicaţii, cu primul curs, publicat ca profesor la Institutul Politehnic din Bucureşti încă din 1925 şi în reţele telefonice, primul studiu pentru ţara noastră fiind elaborat în 1945.

— *Augustin Maior* (1882—1964), inventatorul şi experimentatorul telefoniei multiple care reuşeşte să transmită, în 1906, cinci convorbiri pe un singur circuit telefonic, pe o linie de 15 km, rezultat publicat în 1907.

— *Tudor Tănăsescu* (1901—1961), profesor la Institutul Politehnic Bucureşti, unde a organizat laboratoarele de radioelectronică şi apoi a contribuit la înfiinţarea Facultăţii de Electronică, poate fi considerat întemeietorul şcolii româneşti moderne de electronică. A publicat primele cursuri de radiocomunicaţii (1930), de tuburi şi circuite electronice (1955) şi apoi de tranzistoare (1961). A dezvoltat cercetări în domeniul circuitelor electronice (amplificatoare de putere clasa C, stabilitatea circuitelor şi oscilatoarelor), al electronicii industriale şi nucleare, al automaticii. Profesorul *Tudor Tănăsescu* a fost primul doctor în electronică din România (1940) şi a scris primul studiu din ţara noastră despre calculatoarele electronice.

— *Gheorghe Cartianu* (1907—1982), remarcabil profesor la Institutul Politehnic din Bucureşti, membru corespondent al Academiei din 1964, este considerat creatorul şcolii româneşti moderne de radiotehnică şi radiocomunicaţii. A fost asistent al profesorului *Tudor Tănăsescu* din 1934 şi ulterior, timp de 25 de ani începînd din 1952, şeful catedrei de radiocomunicaţii din cadrul Facultăţii de Electronică şi Telecomunicaţii din Bucureşti. Profesorul *Gheorghe Cartianu* a efectuat cercetări teoretice şi experimentale, privind mai ales modulaţia de frecvenţă, comunicaţiile, circuitele electronice şi cibernetica.

În ultimii 25—30 de ani, profesori şi ingineri de mare prestigiu au condus în continuare activităţile de cercetare şi didactice din domeniul electronicii, privind dispozitivele electronice, mai ales dispozitivele semiconductoare şi circuitele integrate, electronica funcţională, modelarea şi simularea dispozitivelor semiconductoare, circuitele cu impulsuri, automatele secvenţiale, sinteza reţelelor electrice, electronica profesională (aparate), radioelectronica, antenele, electroacustica, televiziunea, teoria şi aplicaţiile transmisiunii informaţiilor.

Dezvoltarea electronicii în ţara noastră trebuie privită şi în corelare cu rezultatele teoretice şi experimentale remarcabile, obţinute în domeniile înrudite ale fizicii, electrotehnicii şi automaticii.

Primele institute de cercetare pentru electronică din România au fost înfiinţate în 1966 şi 1968, primul avînd ca profil electronica profesională de radiocomunicaţii, aparatura electronică de măsură şi electronică medicală, iar cel de al doilea vizînd componentele electronice.

TEHNICA DE CALCUL ŞI INFORMATICA. Evoluţia tehnicii de calcul şi informaticii în România se înscrie în contextul existenţei şi dezvoltării unor serii importante de contribuţii originale româneşti în domeniile matematicii, ciberneticii, electronicii şi automaticii.

Primele calculatoare electronice din România au fost construite la Bucureşti, Cluj-Napoca şi Timişoara.

În anul 1953 începe elaborarea *primului calculator românesc CIFA-1*, de către un colectiv de specialişti* de la Institutul de Fizică Atomică din Bucureşti. Calculatorul CIFA-1 a fost pus în funcţiune în anul 1957, după care au urmat CIFA-2 (în 1959), CIFA-3 (în 1961) şi CIFA-4 (în 1962). CIFA-1 reprezintă totodată primul calculator elaborat în ţările socialiste după realizările din U.R.S.S., deci o replică rapidă dată de România la numai cîţiva ani de la apariţia primelor calculatoare din generaţia I-a pe plan mondial. De altfel, în colaborare cu inginerii români, inginerii din Bulgaria au construit un calculator similar pe baza documentaţiei calculatorului CIFA-1 şi a specializărilor pentru acest produs, calculator denumit iniţial SEMBAN şi expus la Moscova în 1963, la Expoziţia Naţională a R. P. Bulgaria.

Alt colectiv** de la IFA elaborează în anii 1959—1964, calculatoarele *CIFA-101 (1962) şi CIFA-102 (1964)*, tot din generaţia 1, calculatoare ce au beneficiat de numeroase inovaţii tehnologice şi de arhitectură, CIFA-102 fiind implementat în mai multe centre de calcul.

Apoi, colectivul care se afirmase cu modelele CIFA 1—4, realizează în cadrul IFA şi modelele de calculatoare din generaţia a doua, *CET-500 (1964)* şi *CET-501 (1966)*.

Tot la IFA, s-a dezvoltat în anii '60 un colectiv puternic pentru programe de sistem şi programe aplicative.

În Bucureşti s-au mai realizat şi alte calculatoare electronice numerice din generaţiile 1 şi 2, cît şi calculatoare electronice analogice***.

La Institutul de calcul din Cluj-Napoca, înfiinţat în 1957 de academician Tiberiu Popoviciu, se construieşte în 1959 calculatorul experimental cu relee MARICCA, iar apoi un colectiv de cercetare realizează calculatoarele numerice din generaţia a 2-a, *DACCIC-1 (1961) şi DACCIC-2 (1969)*.

DACCIC-1 a fost primul calculator românesc tranzistorizat, cu memorie internă cu ferite, memorie externă cu tambur şi programabil în limbajul FORTRAN, aparţinînd generaţiei a 2-a.

DACCIC-200, construit în numai doi ani (1967—1969), a fost cel mai rapid calculator românesc (200000 operaţii/s) pînă în anii '70 şi primul care a dispus de un sistem de ope-

---

* Cordonator dr. ing. V. Toma; ** Cordonatori ing. A. Segal şi ing. E. Ciobanu; *** MECAN la Institutul de Energetică al Academiei (coordonator prof. V. M. Popov), MAC-1 la Institutul Politehnic Bucureşti (colectiv dr. ing. A. Petrescu, ing. P. Dimo şi ing. I. Şipoş) şi unele modele la Academia Militară din Bucureşti.

rare de concepţie proprie. DACCIC-200 aparţinea tehnologic generaţiei a 2-a, dar încorpora numeroase concepte ce se regăseau la generaţia a 3-a, deci din nou o reuşită românească la numai cinci ani după lansarea generaţiei a 3-a, pe plan mondial, în 1964.

La Institutul Politehnic Timişoara, sub coordonarea mai multor cadre didactice, se construiesc *calculatoarele* MECIPT-1 (1962) din generaţia 1 şi MECIPT-2 (1965)* din generaţia a 2-a.

MECIPT-1, cu tuburi electronice, era similar calculatoarelor CIFA, iar MECIPT-2 a fost primul calculator românesc la care s-a aplicat principiul microprogramării. La elaborarea calculatoarelor MECIPT, la perfecţionarea acestora de-a lungul mai multor ani şi programarea lor, au adus contribuţii o serie largă de cadre didactice, cercetători şi studenţi.

La Cluj-Napoca şi Timişoara s-au înfiinţat, în 1970, primele *Centre teritoriale de calcul electronic din ţara noastră.*

1.2. Evoluţia generaţiilor de calculatoare pe plan mondial şi în ţara noastră

Ce evenimente au marcat însă, evoluţia calculatoarelor în lume?

Pe plan mondial, *primul calculator electronic digital, denumit ENIAC* (Electronic Numerical Integrator And Calculator), a fost construit la Universitatea din Pennsylvania, în intervalul 1942—1945.

ENIAC avea în structura sa aproape 20000 de tuburi electronice, ocupa o suprafaţă de 160 m², avea o greutate de cca. 30 de tone şi o viteză de prelucrare a datelor de maximum 5000 de adunări şi scăderi pe secundă.

Cam în aceeaşi perioadă (1939—1944), la Universitatea Harvard se realizase în colaborare cu inginerii de la firma IBM maşina de calcul automată Mark I, dar care utiliza relee şi benzi de hîrtie perforată.

Construirea calculatorului electronic ENIAC a fost facilitată de elaborarea *maşinii de calcul cu relee MARK I,* de cercetările teoretice ale lui Turing şi de descoperirea circuitelor electronice de numărare, bazate pe tuburi. Despre ENIAC, contemporanii spuneau că „maşina este aşa de mare că de fiecare dată cînd i se dă drumul se ard două tuburi". Introducerea datelor şi a instrucţiunilor se făcea de la 40 de panouri cu fişe, introducerea programului dura cîteva zile, iar verificarea alte cîteva zile. Oricum, programele se rulau de cîte două ori, introducînd din cînd în cînd date de testare pentru găsirea tuburilor arse.

În 1946/1947, *John von Neumann* publica în S.U.A. proiectul primului calculator cu program memorat, cu prelucrare secvenţială a instrucţiunilor şi datelor, memorate împreună, cu aceeaşi formă şi accesibile în acelaşi mod, denumit *EDVAC (Electronic Discrete Variabile Computer).* „Principiul von Neumann" caracterizează aproape toate calculatoarele produse pînă astăzi, acestea fiind numite calculatoare de tip von Neumann**.

Dar prima aplicare a noului principiu din 1946 este datorată rofesorului *M. Wilkes* de la Universitatea Cambridge din Anglia, care a construit, în 1949, primul calculator din lume cu program stocat într-o memorie cu întîrziere — EDSAC (Electronic Delay Storage Automatic Calculator).

Primul calculator comercial a fost livrat de firma UNIVAC la începutul anilor '50.

Specialiştii consideră că în anul 1953 a fost lansată comercial generaţia I de calculatoare electronice, în acelaşi an începînd şi cercetările originale româneşti în acest domeniu, finalizate în 1957, într-o primă etapă, aşa cum s-a arătat mai sus, prin modelul CIFA-1, deci după numai patru-cinci ani!

Prezentăm mai jos cîteva caracteristici care se pot asocia celor cinci generaţii de calculatoare ce s-au succedat pe plan mondial pînă în prezent, cu menţiunea că generaţia a cincea este în curs de elaborare şi finalizare la orizontul anilor '90.

---

* Cu participarea iniţială a specialiştilor D. Kaufmann şi M. Löwenfeld

** Cele mai recente modele, care nu mai respectă acest principiu sînt grupate în *clasa non von Neumann.*

● *Generația I* a cuprins intervalul 1953—1958, (sau după unele criterii extinse perioada 1946—1956), calculatoarele avînd următoarele caracteristici:

— tehnologie cu tuburi electronice;
— arhitectură cu un singur dispozitiv de memorie de mică capacitate și timp de acces mare, de regulă de tip tambur magnetic, utilizat ca memorie internă
— viteza de calcul de cîteva mii de operații / sec. (max. 10 mii operații / sec.)
— programe în cod mașină și în limbaje de asamblare.

● *Generația a 2-a*, 1957/1959-1963/1964, se diferențiază prin:

— tehnologie cu tranzistoare și diode semiconductoare;
— memorie internă, de capacitate sporită (max. 32 koct.) și cu timp de acces mult mai mic, bazată pe inele (toruri mici) de ferită;
— apariția memoriei magnetice externe de tip tambur, disc sau bandă magnetică, în general de mică capacitate;
— viteza de calcul sporită, pînă la cca. 100—200 mii operații / sec.
— îmbunătățirea regimului de lucru al sistemului, astfel încît perifericele (echipamente cu cartele perforate, imprimante rapide) pot lucra simultan cu unitatea centrală;
— apariția primelor sisteme de operare;
— apariția limbajelor de programare de nivel înalt cum sînt: FORTRAN (FORmula TRANslator) lansat de IBM în 1955, cu varianta FORTRAN II în 1958, limbaj destinat calculelor tehnico-științifice, apoi COBOL (COmmon Business Oriented Language) — 1960, pentru aplicații economice cu prelucrări nenumerice și ALGOL (ALGOrithmic Language) — 1960, limbaj algoritmic care facilitează prelucrările numerice, ca și FORTRAN-ul.

● *Generația a 3-a*, după 1964, (care după sursa* citată cuprinde intervalul 1964—1981), se caracterizează prin:

— tehnologie cu circuite integrate;
— memorii interne semiconductoare cu performanțe sporite, cu capacitate pînă la 2 Megaocteți;
— la cele două mari părți distincte ale calculatoarelor din generațiile 1 și 2, hardware (circuite electronice, echipamente) și software (programe de sistem, programe aplicative), se adaugă și a 3-a parte denumită firmware (care conține microprogramele înscrise în memorii fixe);
— discuri magnetice de medie și mare capacitate;
— apariția sistemelor de operare evoluate;
— viteza de calcul apreciabilă, în plaja 0,5—5 MIPS (milioane instrucțiuni pe secundă);
— multiprogramare;
— dezvoltarea limbajelor de programare de nivel foarte înalt: PL/1, PASCAL, LISP, FORTRAN 77 (derivat din FORTRAN IV), BASIC, C și limbajele grafice, completează gama limbajelor anterioare FORTRAN IV — 1962, FORTRAN standard — 1966 și ALGOL-68. Limbajul PASCAL-1971, cu o serie de versiuni ulterioare, facilitează aplicarea principiilor programării structurate. Limbajul LISP-1960, (LISt Processing Language) cu toate dezvoltările care au urmat, este orientat pe prelucrarea datelor reprezentate prin liste, fiind primul limbaj al domeniului inteligenței artificiale. Limbajul BASIC-1975 (Beginner's All-purpose Symbolic Instruction Code) este cel mai răspîndit limbaj interactiv.

● *Generația 3,5—4*, în perioada 1982—1989, implică o serie de progrese tehnologice și arhitecturale:

— circuite integrate pe scară largă și foarte largă (ajungînd la 1 milion de tranzistoare per circuit integrat);
— memorii interne cu circuite integrate, avînd o mare capacitate de memorare (8 Moct.) și un timp redus de acces;
— memorii externe perfecționate, care permit reducerea timpului de acces la informațiile stocate pe suporturi magnetice, unitățile de discuri magnetice devenind dominante în majoritatea configurațiilor (totodată apar discurile optice de mare capacitate);
— sisteme de operare perfecționate, orientate pe lucrul cu discurile magnetice;
— creșterea ponderilor valorice pentru software și firmware față de hardware;
— creșterea fiabilității sistemelor;
— viteza de prelucrare crește sensibil, ajungînd pînă la 30 MIPS;

* Mihai Drăgănescu — „Informatica și societatea", București, 1987.

— generalizarea teleprelucrării, modului de lucru în regim interactiv de la terminale, dezvoltarea facilităților de (inter)conectare în rețele de calculatoare;
— creșterea ponderii implementărilor de sisteme de gestiune a bazelor de date;
— dezvoltarea unor noi limbaje de programare numite concurente, cum sînt: PASCAL CONCURENT (1975), MODULA-2 (1979), ADA (1980) și EDISON (1980), derivînd din PASCAL.

Principiul prelucrării concurente presupune existența simultană a mai multor programe active (sau părți, module, „sarcini") din program, care solicitînd aceleași resurse de calcul, pot fi executate în paralel, ținînd seama de interacțiunile — colaborarea prin comunicare și sincronizare reciprocă — dintre ele.

Limbajul ADA este un limbaj algoritmic modern, care include facilitățile clasice de tip PASCAL, oferind posibilități noi de definire a tipurilor de date corelat cu aplicația, de simplificare a structurilor de control, de modularizare a programelor și de tratare a aplicațiilor în timp real.

Un nou limbaj, influențat de LISP și diferit de limbajele convenționale îl reprezintă FORTH (versiunile 1979 și 1983).

● *Generația a 5-a* de calculatoare, a cărei elaborare a început după 1980, fiind prevăzute pe plan mondial obiective eșalonate pînă la începutul anilor 1990, se distinge substanțial de celelalte generații, urmărind nu atît creșterea vitezei de calcul numeric (pînă la miliarde de operații / sec.), ci mai ales dezvoltarea capacității calculatoarelor de a efectua raționamente, într-un mod apropiat de modul în care raționează experții umani, manipulînd baze de cunoștințe.

În acest sens, dezvoltarea deosebită a domeniului inteligenței artificiale, a programării logice (limbajul PROLOG), corelat cu arhitecturile de tip paralel (prin care se renunță la conceptele von Neumann) și progresele tehnologice impresionante care se întrevăd în vecinătatea anului 1990 (circuite VLSI — cu integrare pe scară foarte largă, cu peste 10 milioane de elemente active pe pastila de siliciu, inclusiv circuite tridimensionale, superdiscuri magnetice și optice cu capacități de memorare de ordinul Gigaocteților și timpi reduși de acces, la costuri tot mai mici), vor permite construirea calculatoarelor mari, mini sau de tip personal, din generația a 5-a, calculatoare suport ale sistemelor expert pentru cele mai variate domenii social-economice. Vor fi implementate limbaje naturale și vor exista facilități pentru dialog prin voce și recunoaștere de forme.

În prezent, au loc pe plan mondial experimente vizînd realizarea calculatoarelor începînd cu generația a 6-a, pe baza progreselor determinante obținute în opto-electronică și mai ales în bioelectronică*.

Ținînd seama de criteriile de mai sus, se prezintă comparativ, în tabelul 1.1, pentru prima dată în acest mod, principalele caracteristici ale cîtorva modele reprezentative de calculatoare în România în perioada 1957—1969 și ale modelelor anilor 1970—1989, evidențiind astfel evoluția generațiilor de calculatoare în țara noastră. Iată o distribuire aproape completă a calculatoarelor românești pe generații:

— generația 1: CIFA 1 ÷ 4, CIFA 101/102, MARICCA, MECIPT-1
— generația a 2-a: CET 500/501, MECIPT-2, DACCIC-1, 2 și 200
— generația a 3-a: FELIX C-256, FELIX C-32, FELIX C-512/1024
— generația 3,5: seriile INDEPENDENT și CORAL, FELIX C-5000.

Microcalculatoarele evoluate românești bazate pe structuri cu microprocesor de 16 biți aparțin, conform criteriilor de mai sus, generației 3,5 — 4 și vor fi prezentate distinct, în capitolul 25.

Din tabelul 1.1, se constată evoluția deosebit de dinamică a performanțelor calculatoarelor românești în cei 30 de ani, cu o rată similară celei medii pe plan mondial și anume de la calculatoare cu tuburi electronice (cu o viteză de prelucrare de 50—2000 operații/s., memorie internă mică de tip tambur cu 2—16 kiloocteți, lector lent de bandă perforată, 16/32 instrucțiuni, programare în cod mașină și o putere consumată de 5 kW), la minicalculatoarele actuale cu circuite integrate de memorare și microprocesoare (cu o viteză de prelucrare de 0,5—2,5 milioane de operații/sec., memorie internă de 1—4 Megaocteți, periferice și terminale, inclusiv unități de discuri de mare capacitate, 200 de instrucțiuni, sistem de operare evoluat, compilatoare pentru limbaje de nivel înalt, limbaj de asamblare, pachete extinse de programe aplicative, sisteme de gestiune a bazelor de date), minicalculatorul avînd la aceste performanțe, numai jumătate din puterea consumată la modelele din prima generație.

---

* Mihai Drăgănescu — „Actualitate și perspectivă în bioelectronică și electronică moleculară", în Buletinul Român de Informatică și Tehnică de Calcul, vol. VIII, nr. 4 din 1987, buletin editat de I.T.C.I. București.

EVOLUŢIA GENERAŢIILOR DE CALCULATOARE

GENERAŢIA ŞI MODELUL CALCULATORULUI	GENERAŢIA 1			GENERAŢIA 2
	CIFA 1/4	CIFA 101/102	CET 500/501	MECIPT 2
ANUL LANSĂRII	1957/1962	1962/1964	1964/1966	1965
NUMĂR MINIM DE SISTEME PRODUSE	4	1/5	1/1	1
TEHNOLOGIA — circuite integrate (buc.)	—	—	—	—
— tranzistoare (buc.)	—	—	2700/4000	10 000
— diode semiconductoare (buc.)	2500	3000	1900/3000	da
— tuburi electronice (buc.)	800	350	—	—
VITEZA DE CALCUL	50 op/s	50—2000 op/s	5 KIPS/12 KIPS	1000—2000 op/s
MEMORIA INTERNĂ — tipul — capacitatea	tambur 50 rot/s 512 cuvinte × ×4 oct	tambur 50 rot/s 4 K cuv. × ×4 oct	ferite 1 K cuv × ×37 biţi	tambur/ferite 4 K cuv × ×38 biţi
ECHIPAMENTE PERIFERICE PRINCIPALE	LBP 15 car/s MS 8 car/s	LBP 15/100 car/s MS 8 car/s	LBP 100/300 car/s MS 8 car/s IMP 5 linii × ×160 car/s	LBP MS, IMP
NUMĂR DE INSTRUCŢIUNI	16	32	32/64	100
LUNGIMEA CUVÎNTULUI (biţi)	31	32	37	38
MODUL DE PRELUCRARE AL CUVÎNTULUI/ TIPUL UNITĂŢII CENTRALE	paralel	serie	paralel	paralel/microprogramat
SISTEM DE OPERARE	—	—	—	monitor INEX conversie date
LIMBAJE DE PROGRAMARE	cod maşină	cod maşină	cod maşină	ASAMBLOR
SISTEME DE GESTIUNE A BAZELOR DE DATE	—	—	—	—
PUTERE CONSUMATĂ	5 KW	1 KW	0,2/0,5 KW	2 KW

— LBP — Lector de bandă perforată
— MS — Maşina de scris
— IMP — Imprimantă
— LC — Lector de cartele
— UDM — Unitate de discuri magnetice
— UBM — Unitate de bandă magnetică

— op/s — operaţii/s
— rot/s — rotaţii/s
— car/s — caractere/s

Tabelul 1.1.

ELECTRONICE ÎN ROMANIA

	GENERAȚIA 3		GENERAȚIA 3,5	
DACCIC—200	FELIX C-256	FELIX C-512/1024	FELIX-5000	INDEPENDENT/ CORAL
1969	1970	1976/1977	1987	1982—1989
1	300	200	10	∼500/500
 10.000 da —	$\times 10^3$ c.i. standard da da —	$\times 10^4$ c.i. standard da da —	1,5$\times 10^3$ c.i. TTL, MOS/LSI și μP da da —	10^3 c.i. TTL, MOS/LSI și μP da da —
200 KIPS	170 KIPS	300 KIPS	300 KIPS	0,5—2,5 MIPS
ferite 8÷32 K cuv× ×25 biți	ferite 256 K oct	ferite/MOS 512/1024 K oct	MOS 4 M oct	MOS 1÷4 M oct
tambur, LBP (opțional UMB) IMP, LC	×UDM 29/58 M oct ×UBM IMP, LC	×UDM 58 M oct ×UBM IMP, LC	×UDM 58/200 M oct ×UBM IMP Terminale	×UDM 58/200 M oct ×UBM IMP Terminale
64	102	117	117	204
24	32	32	32	16/32
paralel	paralel/cablat	paralel/cablat	paralel/microprogramat	paralel/microprogramat
monitor pentru gestiune periferice, tratare întreruperi și multiprogramare	SIRIS-3	SIRIS-3 HELIOS	SIRIS-3 HELIOS	MIX-PLUS SISTEM U(UNIX) MIX/VMS
FORTRAN 2 ASAMBLOARE (PAS și MOL)	FORTRAN COBOL ASSIRIS	FORTRAN IV, COBOL, PASCAL, PL/1, LISP, ASSIRIS	FORTRAN IV, COBOL, PASCAL, PL/1, LISP, ASSIRIS	FORTRAN-77, COBOL, PASCAL, BASIC, ADA, C, LISP, PROLOG, MACRO
—	SOCRATE, MISTRAL	SOCRATE, MISTRAL	SOCRATE, MISTRAL	LEDA, ARGUS, TRANS, RECOL, MIDAS, FOCUS, SOCRATE MINI
3 KW	15 KW	20 KW	15 KW	2÷3 KW

KIPS — 10^3 instrucțiuni/s
MIPS — 10^6 instrucțiuni/s
K oct — 10^3 octeți
M oct — 10^6 octeți
K cuv — 10^3 cuvinte

Tabelul 1.2.

EVOLUŢIA GENERAŢIILOR DE CALCULATOARE ŞI TELECOMUNICAŢIILOR PE PLAN MONDIAL

Generaţia Perioada	1 1946—56	2 1957—63	3 1964—81	3,5÷4 1982—89	5 1990—
1	2	3	4	5	6
Hardware calculatoare	Relee Tuburi electronice Tambur magnetic Tub catodic	Tranzistoare Memorii cu ferite	Circuite integrate Memorii semiconductoare Microprocesoare Discuri magnetice Minicalculatoare	Sisteme distribuite VLSI Discuri optice Microcalculatoare de 16/32 biţi Minisupercalculatoare Supercalculatoare	Tehnici evoluate de împachetare şi interconectare Integrare pe scara ultra largă (ULSI) Proiectarea circuitelor integrate 3-D Tehnologii Ga-As şi Josephson Componente optice Arhitecturi paralele pentru prelucrarea inferenţelor
Software calculatoare	Programe cablate Cod maşină Autocod	Limbaje de nivel înalt (ALGOL, FORTRAN, etc)	Limbaje de nivel foarte înalt Sisteme de operare orientate pe limbajul Pascal Programare structurată LISP Timp partajat Grafică pe calculator Baze de date	ADA Pachete de programe de largă utilizare Sisteme expert Limbaje orientate pe obiecte Baze de date relaţionale	Limbaje concurente Programare funcţională Prelucrare simbolică (limbaje naturale, recunoaşterea formelor: vedere/vorbire, planificare) PROLOG Baze de cunoştinţe Sisteme expert evoluate

Tabel 1.2. (continuare)

1	2	3	4	5	6
Exemple de calculatoare	Eniac, Edvac, Univac, IBM 650 CIFA 1 ÷ 4 CIFA 101/102 MARICCA MECIPT-1	NCR 501 IBM 7094 CDC 6600 DACICC-1/2 CET 500/501 MECIPT-2 DACICC-200	IBM360-370 PDP-11 Spectra 70 Honeywell 200 Cray-1 Illiac-IV Cyber-205 RIAD-1* RIAD-2* Felix C-256 Felix C-32 Felix C-512/1024 Independent 100 Independent 102F Coral 4001 Coral 4030	IBM 43XX VAX-11/7XX IBM-308X Amdhal 80 RIAD-3* Coral 4021 Independent 106 Felix C-5000 Coral 8730	Proiect al Japoniei și alte proiecte noi elaborate în unele țări sau grupuri de țări
Tehnologia telecomunicațiilor	Telefon Teletype	Transmisiuni numerice Modulație în coduri de impulsuri	Comunicații prin satelit Microunde Rețele Fibre optice Comutare de pachete	Rețele integrate de comunicații numerice (digitale)	Dezvoltarea extensivă a sistemelor distribuite, modularitate Fuzionarea tehnologiilor comunicațiilor și calculatoarelor
Performanțele calculatoarelor	Memorii de 2 koct Viteza de operare: 10000 instr./s	Memorii de 32 koct Viteza de operare: 200000 instr./s	Memorii de 2 Moct Viteza de operare: 5 mil. instr./s	Memorii de 8 Moct Viteza de operare: 30 mil. instr./s	Viteza de operare: 1 Giga instr./s ÷ 1000 Giga instr./s (1 G = 1000 Mil.)

* RIAD 1, 2 și 3 reprezintă familii de calculatoare compatibile, elaborate și produse în unele țări membre ale CAER

În tabelul 1.2. se prezintă evoluțiile generațiilor de calculatoare și telecomunicațiilor pe plan mondial, ilustrînd caracteristicile tehnice pentru cele 5 generații definite pînă în prezent, referitor la hardware (tehnologia calculatoarelor, configurații), software (limbaje, sisteme de operare, aplicații), telecomunicații și performanțele calculatoarelor. Alături de modelele de calculatoare românești sînt incluse și cîteva exemple de calculatoare și familii de calculatoare cunoscute pe plan mondial.

1.3. Industria românească de calculatoare electronice. Informatica

Gînditori și economiști români au militat pentru crearea unei industrii naționale, apoi pentru industrializarea țării, încă de la finele secolului trecut și de la începutul acestui secol, creîndu-se o tradiție progresistă a gîndirii românești.

Electronica, domeniu industrial ce marchează profund societatea omenească în acest secol, se poate considera că s-a dezvoltat în România cu precădere după cel de-al doilea război mondial, în două etape.

În *prima etapă*, pînă în cea de-a doua jumătate a deceniului șapte, asistăm la edificarea unei industrii orientate mai ales pe producția de bunuri de larg consum: receptoare radio, TV etc.

În *etapa următoare* au fost realizate de către stat o serie de investiții care, în principiu, au avut ca scop realizarea unei baze de componente electronice pasive și active (rezistori, condensatori, tranzistori, circuite integrate ș.a.), cît și implementarea unui transfer de tehnologie ce a vizat producția de aparatură electronică de automatizare și tehnică de calcul.

La început, pe bază de licențe, au fost asimilate familii de echipamente de automatizare, cît și echipamente de tehnică de calcul de capacitate mică și medie. S-a urmărit, astfel, atît echiparea industriei cu aparatură de automatizare, cît și introducerea unei evidențe economice, în condițiile încercării de realizare a unui sistem informațional economic, a folosirii calculatoarelor în proiectare, transporturi etc.

Deși, în mare, deciziile privind susținerea unor asemenea domenii, la prima vedere, par a fi corecte sub aspect economic, ele au fost într-o măsură hotărîtoare influențate de factorul politic. Aceasta a condus la soluții neeconomice, fără a se ține seamă de tendințele pe plan mondial și de studiile elaborate de specialiști, România plătind, în condițiile unor perspective limitate, un preț deosebit de mare pentru tot ceea ce a realizat.

În deceniul opt alocațiile valutare au fost din ce în ce mai reduse, fapt care a influențat negativ calitatea și competitivitatea produselor românești. Hotărîrile au fost luate de factorii politici, în atmosfera conducerii întregii economii pe bază de indicații, orientări și soluții aberante.

În continuare se prezintă unele date privind evoluția domeniului tehnicii de calcul în țara noastră.

Trebuie arătat că în *învățămîntul superior*, chiar din 1972, s-a revenit asupra nomenclatorului de specialități, specialitatea calculatoare dispărînd. Această decizie aberantă a influențat în mod negativ pregătirea specialiștilor în domeniul tehnicii de calcul, ca de altfel și în alte specialități de vîrf. Cadrele didactice au fost obligate să recurgă la numeroase artificii, pentru asigurarea predării cunoștințelor de specialitate, în condițiile unui trunchi comun de discipline cu caracter general și lateral, care ocupa mai bine de 65—70 la sută din întregul buget de timp al planului de învățămînt.

La 1 noiembrie 1967 a fost înființată *Comisia Guvernamentală pentru dotarea cu echipamente de calcul și automatizarea prelucrării datelor* (desființată după cîțiva ani), iar în trimestrul I al anului 1968, s-a înființat *Institutul de Cercetare și Proiectare pentru Utilaj Electronic de Calcul (ICPUEC)* prin reunirea principalelor grupuri de cercetare din București, Cluj-Napoca și Timișoara.

În anii 1968—1970, s-au înființat Institutul Central de Informatică (ICI), Întreprinderea pentru Întreținerea și Repararea Utilajelor de Calcul (IIRUC), Întreprinderea de Calculatoare Electronice (ICE), Întreprinderea de comerț exterior de specialitate s-a extins profilul IPRS. — Băneasa în domeniul producției de circuite integrate T.T.L.*

Apoi, în perioada 1973—1975 s-au constituit *Societatea Mixtă ROMCONTROL DATA (RCD) pentru unele tipuri de echipamente periferice, Fabrica de Memorii Electronice și Componente pentru Tehnica de Calcul (FMECTC — Timișoara) și Întreprinderea de Echipamente Periferice (IEPER)*, fiind astfel conturată, pînă în 1975, structura de cercetare, producție și service a domeniului tehnicii de calcul și informaticii și consolidată într-o mică măsură baza inițială pentru industria de componente.

Anii 1967—1975 au constituit perioada de organizare a informaticii prin concentrarea acestei activități în centrele de calcul teritoriale, departamentale și uzinale, înființarea unor centre de calcul noi și dotarea acestora avînd loc pe măsura dezvoltării producției proprii.

După 1980 Microelectronica București a început să producă circuite integrate tip MOS**

Treptat au fost înființate și dezvoltate o serie de unități de pregătire și specializare a cadrelor, ori au primit astfel de sarcini unele unități existente, menționînd astfel: *CEPECA — Centrul de perfecționare a cadrelor, Centrul de Calcul economic și cibernetică economică de la ASE București, Centrul de calcul al Universității București, secțiile de specialitate în diferite facultăți ale institutelor de învățămînt superior din București, Timișoara, Cluj-Napoca și Iași*, respectiv liceele cu profil de informatică din București, Cluj-Napoca, Iași, Timișoara, Brașov și Petroșani.

De menționat că în anul 1985 a fost înființat *Institutul de Cercetare Științifică și Inginerie Tehnologică pentru Tehnica de Calcul și Informatică* (ITCI — București) prin unirea formală a resurselor umane și materiale existente în domeniul cercetărilor de profil, ceea ce a condus în ianuarie 1990 la adoptarea deciziei juste de descentralizare prin organizarea a trei institute, cu domenii de activitate clar specificate.

*** T.T.L. = logică tranzistor — tranzistor**

**** MOS = metal — oxid — semiconductor**

Toate aceste întreprinderi şi institute de profil îşi desfăşoară activitatea în cadrul *Ministerului Industriei Electrotehnicii, Electronicii şi Informaticii* (nou înfiinţat după Revoluţia din Decembrie 1989), în afară de Institutul de Cercetări pentru Informatică, centrele teritoriale de calcul şi alte unităţi ce depind de noua Comisie Naţională pentru Informatică, dependentă direct de Consiliul de Miniştri.

Dintre evenimentele şi tendinţele apărute în aceşti 20 de ani în tehnica de calcul românească, menţionăm cîteva mai importante (suplimentar faţă de cele date în tabelul 1.3, în care se ilustrează evoluţia produselor noi din domeniile electronicii şi tehnicii de clacul) şi anume:

— creşterea lentă a producţiei fizice de calculatoare, înnoirea bazată numai pe cercetări proprii, a structurii producţiei destinate atît dotării interne, cît şi exportului;

— organizarea şi extinderea Bibliotecii Naţionale de Programe la Institutul Central de Informatică, dezvoltarea reţelei de centre de calcul teritoriale şi uzinale;

— transformarea ţării noastre în ţară exportatoare de tehnică de calcul începînd cu anul 1978, de cînd s-au exportat peste 200 de minicalculatoare INDEPENDENT şi CORAL, în ţări socialiste şi capitaliste, inclusiv ca sisteme de aplicaţii la cheie;

— dezvoltarea, în special în ultimii 5 ani, a unei familii de modele de calculatoare personale, inclusiv personal-profesionale, fabricate însă în serii total insuficiente faţă de cerinţe.

La realizările industriei de tehnică de calcul şi informaticii româneşti, (pe care nu le vom prezenta în detaliu în această carte), în contextul dat, *un aport însemnat l-a avut şi îl are în continuare învăţămîntul superior* şi, în principal, Institutul Politehnic Bucureşti prin Facultatea de Automatică (Catedra de Calculatoare) şi Facultatea de Electronică.

Profesori şi şefi de lucrări de la Catedra de Calculatoare din IPB au contribuit pe baza unor iniţiative proprii, la dezvoltarea unor domenii, la proiectarea şi asimilarea unor produse noi, cum sînt:

— calculatoare personale şi personal-profesionale (de 8 şi 16 biţi) compatibile cu alte tipuri larg răspîndite pe plan mondial (asimilate în colaborare cu Întreprinderea de Calculatoare Electronice);
— terminale inteligente, display-uri grafice (în curs de asimilare la Întreprinderea de Calculatoare Electronice);
— inteligenţă artificială şi sisteme expert în diferite domenii (proiectare de maşini, medicină etc.);
— baze de date relaţionale;
— compilatoare noi şi instrumente software;
— proiectarea circuitelor VLSI* şi a noilor arhitecturi de calculatoare;

* VLSI = circuite integrate pe scară foarte largă (very large scale integration)

Tabelul 1.3

EVOLUŢIA ELECTRONICII ŞI TEHNICII DE CALCUL ÎN ROMÂNIA ILUSTRATĂ PRIN CÎTEVA PRODUSE REPREZENTATIVE

Anul orientativ al lansării produsului	Tipul
Bunuri de larg consum	
1949	Radioreceptor cu tuburi electronice
1960/1961	Radioreceptoare tranzistorizate
1961	Televizoare alb/negru cu tuburi electronice
1972/1974	Televizoare alb/negru cu tranzistoare şi circuite integrate
1984	Televizoare color cu circuite integrate
Componente electronice active	
1961	Tranzistoare cu germaniu
1969/1970	Circuite integrate TTL
1980/1981	Circuite integrate MOS şi CMOS
1985/1986	Microprocesoare de 8 biţi şi memorii integrate în tehnologie MOS
Calculatoare electronice	
1957	CIFA-1, primul calculator electronic paralel cu tuburi
1962	CIFA-101 primul calculator electronic serial cu tuburi
1964	CET 500, primul calculator electronic tranzistorizat
1969	DACCIC 200, primul calculator electronic tranzistorizat de mare viteză
1969	Calculatoare de birou respectiv maşini de facturat şi contabilizat tranzistorizate pe baza licenţei FRIDEN-SINGER
1970	FELIX C-256, primul calculator electronic cu circuite integrate produs în serie pe baza licenţei firmei CII — Franţa (modelul IRIS-50)
1973	Blocuri şi module de memorie internă cu ferite
1974	FELIX C-32, primul calculator electronic de concepţie proprie, dezvoltat pe baza tehnologiei modelului C 256
1975/1976	Calculatoare de facturat şi contabilizat FC-16, 32, 64 şi 128, cu circuite integrate, dezvoltate prin concepţie proprie pe baza modelelor anterioare
1976/1977	FELIX C-512/1024, al doilea calculator electronic de concepţie proprie dezvoltat pe baza tehnologiei modelului C-256, dar avînd performanţe net superioare, viteza de prelucrare aproape dublă şi capacitatea memoriei interne mărită de 2÷4 ori
1976/1977/1980	Microcalculatoare româneşti de 8 biţi (bazate pe microprocesoarele 8008 şi 8080) MC-8, MC-80, M-18, M-118
1977/1978 cu modele modernizate în 1982/1985	INDEPENDENT şi CORAL, primele minicalculatoare de 16 biţi de concepţie proprie, compatibile la nivel de utilizator cu seria PDP-11 larg răspîndită pe plan mondial (I-100, 102F, 106 şi CORAL 4011, 4030, 4021, 4015)
1983/1986	Microcalculatoare de 8 biţi bazate pe microprocesorul Z-80 cu memorie externă tip disc flexibil (M-118 B, Junior, CUB-Z) şi cele cu minicasetă (aMIC, PRAE, HC85, TIM-S, COBRA); FELIX-PC, microcalculator profesional personal de 16 biţi, compatibil cu IBM-PC şi bazat pe microprocesorul INTEL 8086/8088
1987	FELIX-5000 calculator de medie capacitate de 32 biţi, microprogramat, cu o capacitate a memoriei interne crescută de 4 ori faţă de modelul anterior FELIX C-1024, avînd o tehnologie pe scară largă, (unitatea centrală cu numai 7 plăci imprimate) şi dimensiuni de minicalculator, păstrînd compatibilitatea cu seria FELIX
1987/1988	Junior XT, microcalculator profesional personal de 16 biţi, compatibil cu IBM PC/XT
1988/1989	În cadrul familiilor CORAL (modelul 8730) şi INDEPENDENT sînt elaborate tipurile de 32 de biţi, primul fiind deja asimilat în producţie

Notă: Evoluţia echipamentelor periferice, terminalelor, sistemelor de operare şi pachetelor de programe aplicative *nu* se prezintă explicit în acest tabel.

De asemenea, cadrele didactice ale Facultăţii de Electronică din IPB au adus contribuţii însemnate la dezvoltarea unor domenii şi la proiectarea-asimilarea unor produse noi cum sînt:

- minicalculatoare tip CORAL (asimilate în colaborare cu Întreprinderea de Calculatoare Electronice);
- display-uri grafice (produse la IEPER);
- maşini LISP (în curs de asimilare la IEPER);
- electronica funcţională;
- prelucrarea imaginilor;
- modelarea şi simularea dispozitivelor semiconductoare şi circuitelor VLSI.

Contribuţii noi aduc în prezent, în colaborare cu institutele şi întreprinderile de profil, cadre didactice de la Institutul Politehnic Traian Vuia din Timişoara (microcalculatoare, limbaje), Universitatea Braşov (microcalculatoare), Universitatea Bucureşti (simulare), Universitatea Cluj-Napoca (limbaje), Institutul Politehnic din Iaşi (prelucrarea imaginilor), ş.a.m.d.

Astfel, în ultimii ani, în industria românească de tehnică de calcul şi informatică, pe baza colaborării dintre colectivele de specialişti din cercetare, producţie şi învăţămîntul superior, s-au intensificat cercetările, obţinîndu-se rezultate noi în domenii de mare interes, cum sînt:

- calculatoare personale şi personal-profesionale, domeniu la care se vor face referiri de ansamblu şi de detaliu în această carte; de pildă, modelele aMIC şi HC-85, respectiv modelele anterioare MC-8, MC-80, M-18, M-118, Junior şi cele noi, CUB-Z, FELIX-PC şi Junior XT la asimilarea cărora şi-au adus contribuţia cadrele didactice de la Catedra de Calculatoare din IPB şi o serie de specialişti din industrie, apoi modelele PRAE elaborate de colectivul de la ITCI Cluj-Napoca, modelul COBRA proiectat la ITCI Braşov şi TIM-S asimilat la FMECTC — Timişoara în colaborare cu IPT. Modelul aMIC este deja cunoscut cititorilor prin cartea în două volume ce i-a fost dedicată, iar modelul HC-85 se prezintă pe larg în această carte;
- minicalculatoare fiabile, cu productivitate sporită, modelele INDEPENDENT şi CORAL de 16 şi 32 biţi;
- calculatoare de capacitate medie FELIX C-5000 compatibile cu seria FELIX anterioară.

Menţionăm de asemenea şi alte direcţii importante ale cercetărilor din ţara noastră în acest domeniu, cum sînt:

- conceperea sistemului informatic naţional, elaborarea de metode, tehnici, modele şi elemente tipizate destinate realizării componentelor acestuia, precum şi a reţelelor largi şi locale de calculatoare, mini şi microcalculatoare;
- elaborarea sistemelor de operare noi pentru microcalculatoare (sistem U), minicalculatoare (MIX-PLUS) şi calculatoare medii (HELIOS);
- realizarea unor noi compilatoare, programe de sistem şi aplicative;
- realizarea sistemelor de gestiune a bazelor de date (produsele SOCRATE-MINI — de către Centrul teritorial de calcul Constanţa, respectiv ARGUS, LEDA, RECOL, STAR, FOCUS şi TRANS elaborate la ITCI);
- sisteme de programe pentru proiectarea asistată de calculator şi sisteme grafice la cheie pentru aplicaţii în electronică (proiectarea circuitelor larg integrate şi a plăcilor imprimate), mecanică, arhitectură, construcţii, topografie, cartografie;
- sisteme informatice pentru conducerea producţiei, proceselor tehnologice şi gestiune în unităţile economice, precum şi a sistemelor flexibile de fabricaţie;
- inteligenţă artificială, sisteme expert şi baze de cunoştinţe;
- ingineria programării;
- echipamente periferice de tip bandă magnetică cu transfer continuu, ori de tip cititor de cartelă magnetică cu cap magnetic integrat;

În noile condiţii revoluţionare se acordă o atenţie deosebită organizării şi consolidării în ţara noastră a industriei de calculatoare şi programe.

1.4. Aplicaţii ale calculatoarelor în societatea modernă

Cu toate că s-au scurs puţini ani, chiar relativ la istoria civilizaţiei tehnice moderne a omenirii, calculatoarele electronice au pătruns în activitatea şi chiar în viaţa personală pe o scară atît de largă încît ele reprezintă astăzi o resursă indispensabilă a societăţii. Evoluţia lor, atît constructiv cît şi ca utilizare, se află în strînsă interdependenţă, atît cu evoluţia tehnicilor şi tehnologiilor din toate domeniile de activitate socială, cît şi cu evoluţia gîndirii umane privind modul de abordare a dezvoltării diverselor ramuri economice. Calculatoarele au oferit gîndirii umane mijloace noi de investigare, posibilităţi noi de sistematizare a datelor care erau de multe ori ignorate, de analiză a interdependenţei lor, au determinat chiar o organizare superioară, prin simplitate şi limpezime, a modului de abordare a problemelor.

În prima etapă a evoluţiei calculatoarelor (pînă în 1960) cererile de astfel de produse, cu excepţia unor cereri izolate, unde se impuneau din necesităţi obiective, au apărut din motive de modă tehnică, de teama rămînerii în urmă, tradiţia fiind spre sisteme tot mai performante ca putere de calcul. Reacţia celor care au investit sume mari în echipamente şi specialişti de înaltă calificare, capabili să le utilizeze, a fost promptă: ceva trebuia schimbat atît în privinţa maşinilor, la care se dovedea necesar să fie dimensionate nu numai pentru superprobleme, ci şi pentru aplicaţii mai limitate, de zi cu zi, cît şi în privinţa modului de interacţionare cu operatorul, care era de dorit să fie chiar utilizatorul procesului de calcul. Efectul acestei reacţii îl regăsim în jurul nostru *astăzi cînd întîlneşti calculatoare în grădiniţe, şcoli, la domiciliu, în cercetare-proiectare, în conducerea de utilaje, instalaţii şi procese industriale, în investigaţii medicale, în biblioteci, în magazine, în bănci, în aeroporturi, la întreceri sportive, în prognozarea meteorologică, în teledetecţie la prognozarea recoltelor de cereale, în comunicaţii, la controlul zborului navetelor spaţiale.* Ele sînt utilizate direct de oameni a căror profesie aparţine domeniului de aplicaţie şi nu de specialişti în domeniul calculatoarelor.

Unul dintre cele mai populare exemple în această direcţie îl reprezintă jocurile, cum este jocul de ping-pong „jucat" acasă pe televizorul existent. Videojocurile reprezintă prima utilizare de masă a ceea ce se defineşte sub denumirea de „computer graphics", adică crearea şi manipularea de imagini cu ajutorul calculatorului. Un proverb datînd din antichitatea chineză sintetizează importanţa acestui mod de abordare a interacţiunii om-calculator afirmînd că „o imagine valorează cît o mie de cuvinte".

Domeniile de utilizare a calculatoarelor, aşa cum rezultă de mai sus, sînt extrem de numeroase şi în continuă expansiune. În continuare se vor prezenta cîteva direcţii de aplicare.

În cercetarea ştiinţifică modernă, modelarea matematică a fenomenelor şi proceselor fizice reprezintă un instrument extrem de puternic. Oamenii de ştiinţă elaborează modelul matematic al fenomenului în cauză. Pentru ecuaţiile matematice, care stau la baza modelului, se stabilesc metode numerice de rezolvare. Metodele de rezolvare sînt descrise în limbaje de programare, pentru a obţine programele care, în final, se execută pe un calculator dat.

Programele permit obţinerea soluţiilor într-o gamă largă de variaţie a datelor de intrare, a parametrilor proceselor fizice studiate. Se pot analiza situaţii

limită: aducerea sistemelor fizice în starea de avarie, găsirea limitelor de stabilitate a proceselor, studierea unor mărimi care nu pot fi măsurate direct, etc. În acest sens se poate aminti folosirea calculatorului în modelarea fenomenelor întîlnite în instalaţiile termonucleare, în studiul dinamicii aparatelor de zbor supersonice, în studiul sistemelor complexe de transport al energiei electrice, în astrofizică. Calculatorul permite astfel înlocuirea experimentelor în instalaţiile fizice costisitoare, asigurînd reproducerea, prin modelare, a numeroase variante, fără cheltuieli suplimentare.

Un domeniu care a beneficiat din plin de utilizarea calculatorului este cel al *prognozei vremii*, pe baza datelor culese cu ajutorul staţiilor meteorologice, al sateliţilor, etc. Datele sînt prelucrate cu calculatoare de mare capacitate, programate să rezolve ecuaţii cu derivate parţiale, cu numeroase şi variate condiţii la limită şi iniţiale. Rezultatele sînt prezentate sub formă de hărţi sinoptice, cuprinzînd harta teritoriului interesat, curbele izobare, izoterme, etc.

Calculatoarele se folosesc pentru *colectarea şi prelucrarea datelor* în cadrul diverselor experimente. Astfel, prevăzute cu traductoare (echipamente care transformă în semnale electrice — tensiuni sau curenţi — diferite mărimi fizice neelectrice — lungimi, viteze, acceleraţii, debite, presiuni, temperaturi etc.), calculatoarele pot stoca şi prelucra, în timp real, valorile diferiţilor parametri ce caracterizează un proces sau sistem fizic. Rezultatele sînt prezentate în forme adecvate, pe ecranul unui dispozitiv de afişare (display) sau cu ajutorul unui înregistrator (plotter), ceea ce facilitează interpretarea lor de către experimentatori. Un exemplu îl constituie analizoarele de vibraţii pentru motoarele cu ardere internă sau pentru motoarele reactive.

Pe ecran pot fi afişate sub formă grafică amplitudinile semnalelor la frecvenţele de rezonanţă cu valorile numerice corespunzătoare.

Utilizarea calculatoarelor în medicină a căpătat o largă răspîndire. Ele sînt folosite pentru monitorizarea pacienţilor, în sălile de operaţie şi de terapie intensivă, în instalaţiile pentru efectuarea automată a analizelor, în investigaţiile termografice, ecografice şi tomografice (prelucrarea de imagini). Imaginile obţinute cu ajutorul termografului şi al tomografului sînt prelucrate în vederea accentuării contururilor, eliminării unor semnale parazite, calculului suprafeţelor cu o anumită nuanţă de gri şi colorării artificiale (atribuirea unei culori pentru o nuanţă dată de gri), etc.

Un domeniu care utilizează din ce în ce mai mult calculatorul este cel al proiectării — *proiectare asistată de calculator* (PAC). Sistemele de Proiectare Automată (SPA) au căpătat o largă răspîndire în construcţia de maşini, arhitectură, industria lemnului (mobilier), industria uşoară (modele de ţesături, tipare, tricotaje), etc.

Proiectarea automată a matriţelor necesită programe complexe care operează *în trei dimensiuni* (*3 D*), permiţînd afişarea pe ecran a piesei dorite, în diferite proiecţii. Calculatorul furnizează automat banda perforată pentru conducerea maşinii de aşchiere sau electroeroziune, cu comandă numerică. Astfel, creşte productivitatea muncii şi se reduc termenele de execuţie.

Proiectarea asistată de calculator se foloseşte în mod frecvent în *studiul comportării statice şi dinamice a diverselor sisteme, echipamente, maşini*, etc. Astfel, pot fi puse în evidenţă componentele mai solicitate şi se pot lua măsuri pentru creşterea fiabilităţii acestora.

Proiectarea asistată de calculator joacă un rol important în *realizarea de noi substanțe chimice, medicamente*, etc. Pe baza modelelor teoretice, elaborate pentru interacțiunile la nivel molecular, ale diverșilor compuși chimici, se pot proiecta noi compuși, vizualizîndu-se pe ecran structurile moleculare ale acestora, fără a mai efectua experimente de laborator costisitoare și laborioase.

Sistemele de rezervare a locurilor pentru pasageri, în transporturile aeriene sau pe cale ferată, reprezintă unele din importantele aplicații ale calculatoarelor în sfera serviciilor. Agențiile de voiaj sînt prevăzute cu terminale, conectate la distanță, la un calculator central. Acesta din urmă dispune de capacități importante de stocare și regăsire a informațiilor referitoare la cursele de avioane, orarul trenurilor, listele cu pasagerii care au rezervat locurile, numărul locurilor disponibile, etc. Terminalul aflat la agenția de voiaj poate elibera, de asemenea, automat, biletul solicitat la avion sau la tren.

Sistemele de stocare și regăsire a informațiilor și-au găsit importante utilizări *în marile biblioteci și instituții de documentare*. Cărțile, revistele, articolele de specialitate sînt introduse în memoria calculatorului sub forma unor înregistrări care conțin o serie de informații esențiale: numele autorilor, titlul, editura, anul apariției, numărul de figuri, numărul referințelor bibliografice, cuvinte cheie, un scurt rezumat, etc. Pe baza unor cuvinte cheie date, calculatorul selectează, în mod automat, din baza de date, lucrările corespunzătoare, ușurînd astfel, în mod substanțial, munca de cercetare bibliografică.

În ultimii ani o largă răspîndire a găsit *prelucrarea textelor cu calculatorul*. Pe baza unui program special, numit procesor de texte, utilizatorul operează cu calculatorul în maniera în care ar lucra cu o mașină de scris inteligentă. Textul introdus este afișat pe ecran, existînd comenzi speciale pentru inserarea/înlăturarea de cuvinte, fraze sau paragrafe. La sfîrșitul rîndului, automat, se face despărțirea în silabe, respectiv alinierea cuvintelor. Documentele astfel editate sînt stocate pe suport magnetic (disc flexibil, casetă magnetică). Ele pot fi ulterior modificate după cerințe sau extrase la imprimantă, ori de cîte ori este nevoie.

Preocupările constructorilor de calculatoare continuă în direcția simplificării sau naturalizării interacțiunii operator-calculator. În acest sens realizările obținute pe linia sintezei și recunoașterii vorbirii (pînă la 20.000 și respectiv 1.000 de cuvinte) sînt încurajatoare, chiar dacă unii sînt încă sceptici datorită restricțiilor privind ritmul vorbirii. Există anunțuri recente ale unor firme americane privind comercializarea de „secretare computerizate“ care pot redacta corespondența după dictare.

În ultimii ani s-au înregistrat importante progrese pe linia realizării unor sisteme expert, în diverse domenii de activitate: medicină, biologie, geologie, proiectare, întreținere de instalații complexe, etc. Bazate pe calculatoare evoluate înzestrate cu programe adecvate înglobînd principiile inteligenței artificiale, aceste sisteme sînt capabile să manipuleze cunoștințe, folosind reguli de inferență, care permit modelarea raționamentului unor experți umani, în diferite domenii, cu consecințe economice și sociale foarte importante. În cazul *roboților*, problemele ce vizează aspectele de inteligență artificială se referă la faptul că, pentru aceste echipamente nu este suficient să fie capabile să manipuleze obiecte; robotul trebuie să fie capabil să primească și să interpreteze informații legate de mediul înconjurător și să-și adapteze acțiunile sale ținînd cont de aceste informații.

Cu sistemele realizate în sectoarele de proiectare, cu elementele de programare şi conducere a fabricaţiei, cu realizările din domeniul roboţilor, cu maşini prelucrătoare dotate cu calculatoare sau minimum cu comenzi numerice, a apărut ca posibilă o automatizare flexibilă, adaptabilă în timp scurt varietăţii pieselor de fabricat, în scopul obţinerii unei productivităţi cît mai ridicate. Este cunoscut faptul că flexibilitatea şi productivitatea sînt în general două deziderate contradictorii şi de aceea a apărut ca necesară găsirea unei metode de îmbinare optimă a acestora, fără a accentua prea mult pe una în detrimentul celeilalte.

Calculatoarele pot fi interconectate în cadrul unor *reţele locale sau globale* (la nivel de ţară sau continent).

În *reţelele locale*, calculatoarele permit, de pildă, implementarea conducerii operative a unor întreprinderi, instituţii dispuse pe o arie geografică restrînsă.

Reţelele globale conectează, de exemplu, centrele de calcul teritoriale sau calculatoarele plasate la distanţă, în cadrul întreprinderilor diverselor ministere economice.

Reducerea continuă a costurilor componentelor electronice, apariţia microprocesorului, perfecţionarea în general a tehnologiilor electronice, au facilitat *apariţia calculatoarelor personale*. Acestea, în funcţie de tipul microprocesorului, de capacitatea memoriei, de echipamentele periferice folosite pot fi lasate în clasa calculatoarelor personale de uz general, sau în clasa calculatoarelor personale — profesionale (microcalculatoare evoluate tehnologic).

Această dezvoltare tehnologică a calculatoarelor personale, creşterea performanţelor lor pe baza progreselor deosebite ale microelectronicii, le-au transformat într-o resursă importantă pentru rezolvarea unor probleme şi aplicaţii, care pînă în aceşti ultimi 3—5 ani erau adresate minicalculatoarelor şi chiar calculatoarelor medii — mari.

Calculatoarele personale sînt folosite de largi categorii de utilizatori: ingineri, medici, economişti, profesori, tehnicieni, studenţi, arhitecţi, agronomi, muzicieni, elevi, etc.

Calculatoarele informatizează activităţile de rutină, care necesită un mare volum de muncă, permiţînd utilizatorului concentrarea asupra aspectelor creatoare ale muncii sale. Pentru a obţine beneficiile maxime din utilizarea calculatoarelor este necesar ca programarea operării lor să fie efectuată pe baza unor algoritmi corecţi de calcul, fără ambiguităţi, inexactitatăţi, etc. În acest sens se impune din partea utilizatorului o profundă înţelegere a problemei, care este programată spre a fi rezolvată cu calculatorul.

Am început acest subcapitol afirmînd că, în societatea modernă calculatoarele reprezintă o resursă cheie a dezvoltării societăţii; cu fiecare lună, trimestru sau an, ele devin o forţă de producţie esenţială. Toate meseriile sînt practic influenţate de impactul calculatorului şi vor fi în continuare, poate într-un ritm tot mai ridicat. Pregătirea tuturor membrilor activi ai societăţii pentru utilizarea calculatorului reprezintă o condiţie strategică a etapei actuale, această carte încercînd să aducă o contribuţie la punerea în practică a acestei strategii.

Capitolul 2

Caracterizarea și evoluția calculatoarelor personale și microcalculatoarelor

2.1. Ce este un calculator personal?

Cartea aceasta, prima carte scrisă la noi pentru învățarea utilizării unui calculator personal compatibil cu o familie larg răspîndită pe plan mondial, se referă la modelul românesc HC-85, asimilat în 1985 în fabricația de serie la Întreprinderea de calculatoare electronice din București.

Calculatorul HC-85 se înscrie în seria echipamentelor de calcul românești competitive, cu performanțe apreciabile, îndeosebi pentru instruirea asistată de calculator. HC-85 este fiabil, economic și compatibil cu un alt model, larg răspîndit pe plan mondial.

Caracteristicile de mai sus pot fi în general exprimate sub forma unui criteriu sintetic de apreciere, dat de raportul dintre performanță și preț. Raportul este îmbunătățit continuu pe plan mondial, prin creșterea performanțelor (viteza de prelucrare, capacitatea de memorare, facilități pentru un dialog „prietenos" om-mașină, programe eficiente, fiabilitate) și reducerea prețului.

Calculatorul personal este realizat în jurul unui circuit integrat complex, numit microprocesor și care este ajutat în „activitatea" sa de alte circuite, unele mai simple, altele complexe, asigurînd funcțiuni specializate. În ultimii ani au apărut microprocesoare noi, care prelucrează de două ori și chiar de patru ori mai multe unități de informație, în unități de timp de cîteva ori mai mici. Astfel, o dată cu utilizarea microprocesoarelor de 16 și de 32 de biți, a crescut viteza și puterea de prelucrare a datelor.

Totuși, *calculatorul personal HC-85 selectat pentru instruirea elevilor, are un microprocesor clasic, tip Z80A*, de 8 biți, la fel ca alte calculatoare din grupa sa. În țara noastră s-au proiectat și se fabrică și calculatoare cu microprocesoare de 16 biți, evident la un preț de cost mult mai ridicat.

Nu puțini dintre tinerii elevi participanți la cercurile de informatică au o curiozitate științifică demnă de viitori specialiști și își pun o serie de întrebări.

Unii se interesează mai mult de utilizarea calculatorului personal, privit ca un instrument nou, așa cum în alte timpuri au fost abacul sau rigla de calcul. Dorința lor de a deveni utilizatori evoluați este în concordanță cu evoluția folosirii tot mai extinse a calculatoarelor.

Alții se interesează mai ales de „arhitectura" și tehnologia microelectronică pe care se bazează calculatoarele personale și doresc să afle cum funcționează calculatorul și chiar microprocesorul, „creierul și inima" calculatorului personal. Microprocesorul este un circuit integrat cu o zonă activă de cîțiva mm² în care este „imprimată" o schemă echivalentă cu 10 mii sau uneori chiar 300 de mii de tranzistoare, schemă în care cele mai fine trasee au o lățime de 3 microni, 2 microni și mai recent sub 1 micron. În viitor vor coborî sub o jumătate de micron. Și apoi? Iată, notăm și alte întrebări.

Care sînt tendinţele pe plan mondial în acest domeniu? Dar la noi în ţară? Cum se clasifică calculatoarele personale? Unde se încadrează HC-85? Care este durata de viaţă pentru utilizarea calculatoarelor cu microprocesoare de 8 biţi? Care tehnologii ale microelectronicii sînt mai răspîndite? Cum evoluează microprocesoarele? Care limbaj de programare trebuie învăţat mai întîi? Dar specialiştii de azi şi de mîine, ce limbaje au sau vor avea la dispoziţie? Cum se dezvoltă instruirea asistată de calculator pe plan mondial?

Calculatorul cel mai des întîlnit în următorii cîţiva ani, pe masa de lucru a elevilor pionieri, dar şi a elevilor UTC-işti, şi într-o anumită măsură a studenţilor, a inginerilor, a profesorilor, va fi calculatorul HC-85. Vor exista şi alte modele total compatibile cu HC-85, destinate aceloraşi aplicaţii.

Şi totuşi... peste 10—15 ani, elevii aflaţi azi pe băncile şcolii vor deveni muncitori cu înaltă calificare, medici, ingineri, chimişti, fizicieni, matematicieni, profesori. Unii dintre ei vor fi proiectanţi de calculatoare, informaticieni de înaltă competenţă.

Iată de ce, intuind gîndurile multor tineri cititori, includem cîteva pagini despre prezentul şi viitorul calculatoarelor. Vor fi introduse unele noţiuni noi care se vor explica gradat. Se vor da şi trimiteri la o bibliografie selectată şi comentată, care cuprinde atît lucrări de popularizare, cît şi unele lucrări de specialitate existente în biblioteci.

Ce este deci, un *calculator personal*? Cum să definim aceste *microsisteme de echipamente si programe*?

Vom spune simplu, fără a formula o definiţie extinsă, că un calculator personal este interactiv, portabil, are o structură cu unul sau mai multe microprocesoare, o tastatură şi un afişaj de tip TV ori monitor, monocrom sau color, una sau mai multe memorii externe, unul sau mai multe limbaje de programare de nivel înalt, poate cîteva opţiuni (dispozitive şi programe), şi toate acestea la un preţ din ce în ce mai redus.

Dintre caracteristicile de bază, următoarele prezintă o importanţă deosebită şi anume:

— utilizarea individuală interactivă, toate resursele microsistemului fiind la dispoziţia operatorului — programator (exceptînd memoria permanentă, protejată la scriere);

— interfaţa om - maşină „prietenoasă", facilitînd dezvoltarea dialogului şi ghidarea continuă a operatorului;

— facilităţile modelelor profesionale de interconectare în reţele locale sau de cuplare la calculatoare mai mari, permiţînd accesul la baze de date şi cooperarea calculatoarelor la rezolvarea unor probleme de mai mare complexitate.

În continuare, vom considera că un calculator personal (sau, chiar un microcalculator) este alcătuit în principal din mai multe elemente, dispozitive şi programe, selectate dintr-o gamă largă, care cuprinde:

— modulul de bază cu structura logică, cu unul sau mai multe microprocesoare şi alte circuite menţionate mai jos, inclusiv pentru controlul imaginii;

— memoria în care se scrie şi se citeşte orice zonă, pentru programele utilizatorului, respectiv pentru ecran;

— memoria permanentă care se poate doar citi, pentru monitorul de comenzi, interpretor şi programe utilitare;

— circuitele de interfaţă cu echipamentele periferice;

— sursa de alimentare încorporată în calculator sau distinctă;

— dispozitive şi periferice de intrare/ieşire, încorporate în modulul de bază sau distincte, cum sînt: tastatura, dispozitivul de afişaj monocrom/color tip TV sau monitor, dispozitivul de poziţionare cu bilă („mouse") tip penel sau cu manetă („joystick"), ambele cu butoane pentru poziţionarea cursorului pe ecran, miniimprimanta şi trasatorul de curbe (plotter de masă);

— unităţi de memorare externă, încorporate în modulul de bază sau distincte, pentru calculatoare personale uzuale (casetofon) şi pentru tipurile profesionale (unităţi de discuri magnetice flexibile şi/sau unităţi de disc fix sau rigid, tip Winchester, cu capacităţi de memorare de 0,25—1 Moct., respectiv de 10—40 Moct.);

— monitor — dezasamblor, editor, asamblor, interpretor pentru BASIC şi cîteva compilatoare pentru limbaje evoluate (în cazul calculatoarelor personale portabile, de familie, etc.), sau
— interpretor de BASIC extins, eventual sistem de operare, translatoare pentru programe în limbaj de asamblare şi în BASIC, compilatoare pentru limbaje evoluate, respectiv medii de dezvoltare pentru programe noi (în cazul calculatoarelor personale-profesionale);

După descrierea gamei extinse din care se aleg configuraţiile tipice ale calculatoarelor personale şi personal-profesionale, să trecem în revistă cîteva noţiuni utile programatorilor începători, dar mai ales celor care au depăşit fazele iniţiale, noţiuni care se referă la interpretoarele BASIC, compilatoare, monitor, sistem de operare, modul interactiv de lucru şi instruirea asistată de calculator.

Interpretorul BASIC analizează şi execută direct instrucţiune cu instrucţiune programul sursă (scris în *BASIC* de programator). Utilizatorul obţine rezultatele execuţiei programului şi nu un program obiect intermediar în cod maşină, adică în instrucţiunile microprocesorului. Interpretorul asigură un caracter interactiv, dar este în general mai puţin eficient decît compilatorul.

Compilatorul transformă programul sursă (scris într-un limbaj de nivel înalt, de pildă FORTRAN) într-un program obiect echivalent (scris în cod maşină). Aceasta însemna transformarea unui şir de caractere (programul sursă) într-un şir de biţi (programul în cod maşină) şi implică o serie de operaţii:

— analiza lexicală pentru a evidenţia constante, variabile, operatori, cuvinte-cheie.

— analiza sintactică (corecţitudine gramaticală) şi semantică (sensul cuvintelor înlănţuite), pentru a verifica regulile stabilite la definirea sintactică a limbajului;

— generarea intermediară a codului în limbaj de asamblare;

— optimizarea codului pentru reducerea spaţiului de memorare şi a timpului de execuţie;

— generarea codului obiect prin operaţia de asamblare (se traduce limbajul intermediar în cod maşină);

— în toate fazele compilării, enumerate mai sus, au loc două operaţii privind alcătuirea şi gestiunea tabelelor (tabele de simboluri, de constante, de cicluri), respectiv analiza erorilor. Pentru unele compilatoare la care se generează direct codul maşină, operaţia de asamblare nu mai este necesară.

Monitorul reprezintă în cazul calculatoarelor personale, o colecţie de comenzi şi rutine care se pot apela în două moduri: de la tastatură şi prin programele utilizatorului. Monitoarele din această categorie ocupă 1—8 Koct. în memoria permanentă. Monitoarele calculatoarelor personale româneşti introduse în fabricaţie, includ 10—15 comenzi, dintre care menţionăm, de pildă:

— afişarea pe ecran a conţinutului unei zone de memorie;

— afişarea registrelor interne;

— lansarea în execuţie a unui program obiect aflat în memorie.

Un asamblu de componente software Monitor-Asamblor-Editor de texte rezident în memoria permanentă constituie un nucleu de sistem de operare.

Sistemul de operare este, în general, un asamblu de programe care realizează gestiunea resurselor unui sistem de calcul, rezolvă conflictele care apar în alocarea resurselor sau între utilizatori (la microsistemele cu acces multiplu, de la mai multe terminale) şi asigură o utilizare eficientă a echipamentelor şi programelor.

Calculatoarele personal-profesionale produse în țara noastră, pot fi împărțite în două grupe (cărora le corespund sisteme de operare specifice):

— compatibile cu sistemele de operare CP/M pentru microcalculatoare de 8 biți (mult mai răspîndite decît sistemul de operare anterior SFDX).

— compatibile cu sistemele de operare MS-DOS și PC-DOS pentru microcalculatoare de 16 biți (asemănătoare principial cu CP/M).

În aceste două cazuri, încărcarea sistemului de operare în memoria calculatorului se realizează prin transfer de pe suportul magnetic extern.

Modul interactiv, conversațional, de lucru al (micro) calculatorului personal permite utilizatorului să fie în dialog, în contact permanent cu problema dată, astfel încît programele pot fi dezvoltate și corectate mult mai rapid. Un domeniu deosebit de atrăgător pentru instruirea asistată de calculator, îl constituie animația și grafica color interactivă (însoțite de efecte sonore) realizate cu ajutorul unor dispozitive de intrare și a programelor adecvate pentru poziționarea și selecția unui „punct" pe ecran (tip „mouse" sau penel, „joy-stick", creion optic, tastatură funcțională, tabletă grafică, ecran sensibil la atingere, etc.), respectiv cu dispozitive grafice de ieșire (display, miniimprimantă, trasator de curbe).

Limbajele principale utilizate în domeniul calculatoarelor personale sînt: BASIC, LOGO, FORTH.

Pentru cele profesionale, există și alte limbaje evoluate: C, TURBO C, FORTRAN 77, TURBO-PASCAL, MODULA, ADA, LISP și PROLOG.

2.2. Microelectronica (microprocesoare și alte circuite integrate) pe plan mondial

Încă de la mijlocul deceniului trecut, experții atrăgeau atenția asupra faptului că importanța microelectronicii pentru o țară industrializată egalează pe aceea a ingineriei nucleare!

Într-adevăr, întreaga dezvoltare a tehnicii de calcul începînd cu anii '70 se bazează pe *evoluția rapidă a microelectronicii*, pe creșterea spectaculoasă a mărimii seriilor de fabricație, a complexității, vitezei și fiabilității circuitelor integrate, în condițiile reducerii dramatice a consumului de putere și a prețurilor. Tehnica de calcul, microelectronica, informatica, automatizarea, reprezintă pîrghiile cheie pentru creșterea productivității muncii și oferă soluții noi pentru dezvoltarea economică intensivă, în cadrul unor resurse materiale și financiare limitate la anumite niveluri date.

Se poate afirma că, deși valoarea microprocesoarelor și a celorlalte circuite integrate din componența calculatorului este mică, în comparație cu valoarea perifericelor și a programelor de bază și aplicative, totuși rolul acestor circuite complexe și miniaturizate este vital. *Ponderea valorică a circuitelor integrate din totalul echipamentelor de tehnică de calcul reprezintă numai 14%, dar ponderea valorică a producției de circuite integrate din producția de componente semiconductoare este pe plan mondial de circa 85%.* De altfel, toate țările dezvoltate industrial investesc an de an sute de milioane de dolari în dezvoltarea microelectronicii, asigurînd un ritm alert de înnoire a tehnologiilor și producții de mare serie, cu costuri în reducere.

Consumul de circuite integrate are în 1989, conform unor estimări recente, următoarea structură calculată în ponderi valorice (considerînd consumul egal cu producția valorică plus import minus export), pentru 6 țări capitaliste dezvoltate industrial (SUA, Japonia, RFG, Marea Britanie, Franța și Italia):

Tabelul 2.1.

Circuite integrate, total 100%	
din care:	
— Memorii 31%	**— Circuite fabricate la comandă .. 16%**
— Microprocesoare, microcalculatoare într-o singură capsulă și circuite specializate 23%	**— Circuite analogice 19%**
	— Familiile de circuite logice standard 11%

Principalele familii de circuite integrate logice sînt următoarele, date în ordinea descrescătoare a ponderilor valorice, din consumul total de circuite din această grupă, care totalizează doar 11%:

- **TTL Schottky (logica tranzistor-tranzistor)**, cu complexitate medie și viteză mare de lucru (timp redus de propagare a semnalului logic prin circuit, numit și timp de propagare sau comutare), incluzînd și tipurile de circuite TTL Schottky Low Power, deci versiunile cu consum redus de putere;
- **CMOS (logica complementară în tehnologia metal-oxid-semiconductor)**, de mare complexitate și densitate per unitate de arie, cu viteza medie spre mare și consum de putere extrem de redus;
- **ECL (logica cu cuplaj al tranzistoarelor prin emitor)**, de complexitate medie, dar de foarte mare viteză;
- **TTL de mare viteză**, cu complexitate mică spre medie.

Dintre acestea, primele două sînt cel mai mult folosite în domeniul calculatoarelor personale și microcalculatoarelor.

Grupa circuitelor integrate de memorare (31%) include:

- **memoriile cu acces aleator (RAM)**, fiind posibilă scrierea sau citirea datelor în orice zonă a memoriei, cu tipurile dinamice (datele sînt recirculate continuu) și statice. Tipurile de memorii dinamice și cele mai multe din cele statice sînt realizate în diferite versiuni ale tehnologiei MOS, iar memoriile statice au și variante bipolare;
- **memoriile de tip numai-citește (ROM)**, cu mască, unele tipuri fiind programabile (PROM), ori cu posibilitatea de a fi șterse electric (EEPROM), sau cu raze ultraviolete (EPROM). Se mai numesc memorii fixe sau permanente, avînd un conținut fixat de producător, și peste care utilizatorul nu poate înscrie alte date sau programe. Sînt realizate în tehnologiile MOS, CMOS și bipolară;
- **memorii neconvenționale realizate cu tehnologii noi, cum sînt dispozitivele cu cuplaj prin sarcină (CCD) și cele cu bule magnetice (magnetic bubble), ambele avînd însă pînă în prezent o pondere foarte mică.**

Memoriile RAM dinamice (DRAM) și cele fixe de tip EPROM sînt larg utilizate în calculatoarele personale și micro, ca memorii interne pentru date și programe. Dintre memoriile RAM dinamice cu capacități de 64, 256, 1024 și 4096 de kilobiți/circuit integrat, cele mai utilizate sînt cele de 256 kbiți și 1Mbit, care au împreună o pondere valorică de 94% în 1989, din totalul memoriilor RAM destinate utilizării în industria de tehnică de calcul din SUA (cu mențiunea că ritmul de creștere al producțiilor de circuite de 1Mbit este mult mai mare, acest tip tinzînd să depășească 75% în 1989). Prețurile pentru memoriile de 64 și 256 kbiți, chiar pentru cantități mici, au scăzut enorm și au ajuns la un nivel extrem de redus, sub un dolar și respectiv cîțiva dolari, comparativ cu circuitul integrat de 1Mbit care costă cu aproape două ordine de mărime mai mult.

Cu toate acestea circuitul de 1 Mbit cu timp de acces redus la 100 ns este preferat atît din punctul de vedere al performanţei (mai apropiate de aceea a noilor microprocesoare) cît şi din punct de vedere economic (deoarece elimină memoriile RAM statice rapide, folosite pentru a compensa decalajul dintre vitezele de lucru tot mai mari ale microprocesoarelor şi viteza relativ redusă a memoriilor de 64/256 kbiţi).

Cea de a doua grupă de circuite integrate (conform tabelului 2.1) pe care o prezentăm în continuare, este cea mai importantă pentru calculatoarele personale şi microcalculatoare. Această grupă include microprocesoare, microcalculatoare pe o pastilă de siliciu (numite one -chip), circuite integrate specializate pentru comanda memoriei interne, a echipamentelor periferice, pentru prelucrarea semnalelor, sinteza vocii, etc.

Dacă vom considera o subgrupă distinctă compusă din microprocesoare (mP de 8, 16 şi 32 de biţi) şi microcalculatoare one-chip (mC de 4, 8 şi 16 biţi) luate împreună, atunci ponderile în consumul SUA, în 1989, sînt estimate astfel:

Tabelul 2.2

Microprocesoare şi microcalculatoare one-chip, total	Ponderea valorică în 1989, în S.U.A.
	100 %
din care:	
— mP şi mC, de 8 biţi	45 %
— mP şi mC, de 16 biţi	33 %
— mP de 32 biţi	15 %
— mC de 4 biţi	7 %

Din acest tabel rezultă că producţia valorică şi deci consumul valoric de aceste componente profesionale este maxim în plaja performanţelor medii (8 biţi şi 16 biţi), ceea ce este şi mai evident pentru producţia fizică şi consumul fizic exprimate prin numărul de circuite.

S-a exemplificat prin structura consumului SUA de microprocesoare şi microcalculatoare one-chip, deoarece acest consum valoric reprezintă aproape 60% din suma consumurilor similare din alte 5 ţări dezvoltate industrial (Japonia, RFG, Anglia, Franţa şi Italia), iar pe de altă parte, statisticile publicate în literatura de specialitate sînt mai detaliate pentru SUA.

Dacă se analizează livrările fizice (exprimate în numărul microprocesoarelor), atunci se constată că peste 85% din numărul microprocesoarelor livrate au fost tipurile de 8 biţi.

În ce constă de fapt secretul acestei mari durate de viaţă pentru microprocesoarele de 8 biţi, apărute în urmă cu 15—16 ani şi despre care se presupune că nu au atins încă vîrful livrărilor proprii şi vor depăşi 25 de ani de producţie? Cel puţin doi factori au un rol determinant:

— creşterea continuă a performanţelor microprocesorului de 8 biţi de bază, prin încorporarea (integrarea) unor funcţiuni noi de control al memoriei şi al perifericelor, şi prin mărirea vitezei interne de lucru;

— cuplarea arhitecturilor de 8 biţi cu sistemul de operare CP/M, larg răspîndit şi mai simplu decît sistemele de operare Unix sau MS-DOS şi dezvoltarea unui mare număr de programe de aplicaţii compatibile CP/M.

În figurile 2.1 şi 2.2 sînt date evoluţia complexităţii microprocesoarelor, mărimea estimativă a seriilor de producţie pentru circuitele clasice şi de vîrf din familiile microprocesoarelor.

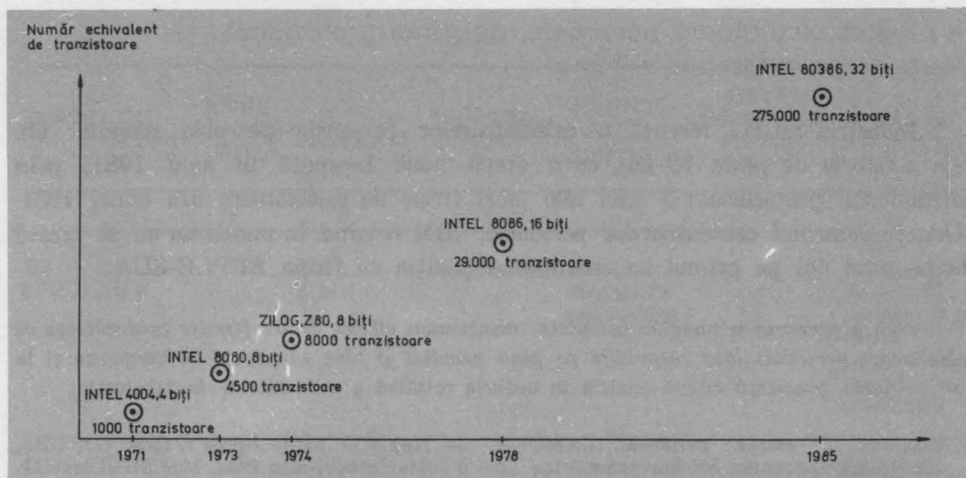


Fig. 2.1. Evoluţia complexităţii microprocesoarelor

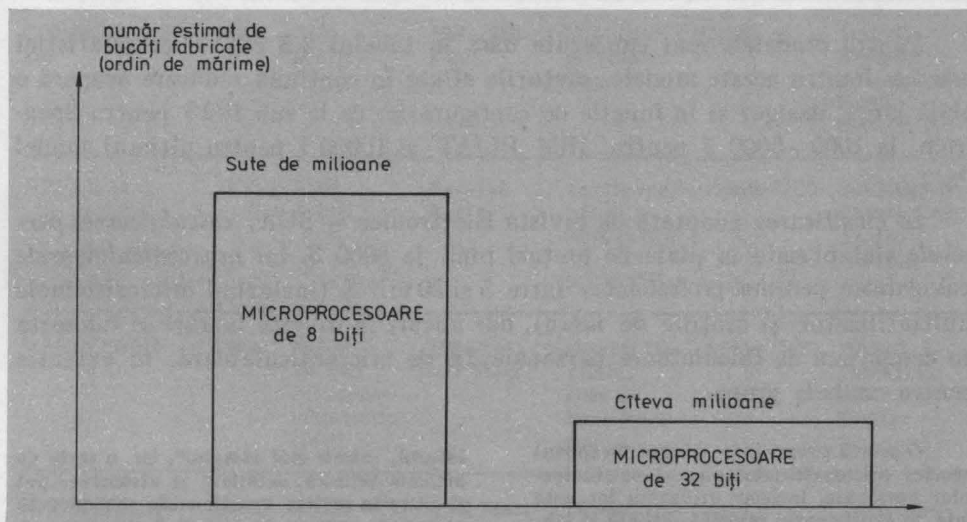


Fig. 2.2. **Producţia mondială de microprocesoare de 8 şi 32 biţi (cantităţi însumate în perioada 1974—1988)**

Iată de ce, colectivul de autori consideră că HC-85, calculatorul personal românesc, bazat pe cunoscutul microprocesor de 8 biţi Z80A (avînd în vedere fiabilitatea sa, extensiile de care dispune, inclusiv pentru lucrul cu unitatea de discuri flexibile, cît şi preţul de cost redus), ca şi celelalte modele româneşti compatibile cu HC-85, reprezintă o bună alegere pentru utilizare largă în şcoli, la cursurile şi la cercurile de informatică pentru elevi, pionieri şi UTC-işti.

2.3. Calculatoare personale, personal-profesionale şi microcalculatoare în lume

Industria relativ recentă a calculatoarelor personale pe plan mondial are deja o istorie de peste 10 ani, cu o etapă nouă începută în anul 1981, prin pătrunderea spectaculoasă a celei mai mari firme de calculatoare din lume, IBM-SUA, în domeniul calculatoarelor personale, IBM reuşind în numai un an să treacă de pe locul doi pe primul loc, schimbînd poziţia cu firma APPLE-SUA.

Fără a prezenta o imagine completă, menţionăm cîteva dintre *firmele producătoare de calculatoare personale larg răspîndite pe plan mondial* şi bine cunoscute de asemenea şi la noi în ţară, precizînd cîteva modele în ordinea relativă a introducerii în fabricaţie:

- Sinclair — Marea Britanie: ZX80, ZX81, ZX Spectrum, ZX Spectrum Plus şi QL;
- Commodore — SUA: PET Commodore, VIC20, Commodore 64, 16, 116, Plus/4, 128 şi PC;
- Apple — SUA: Apple I, II şi III, Lisa, Macintosh, Mac Plus, Mac SE şi Mac II.
- IBM — SUA: IBM PC, PC/XT, PC Jr., PC/AT, PC/RT şi seria PS/2;
- COMPAQ — SUA: DESKPRO 386.

Pentru modelele mai cunoscute dăm în tabelul 2.3 cîteva caracteristici tehnice. Pentru aceste modele, preţurile aflate în continuă reducere acoperă o plajă largă, desigur şi în funcţie de configuraţie, de la sub 100 $ pentru Spectrum, la 1000—5000 $ pentru IBM PC/AT şi 10000 $ pentru ultimul model PS/2.

În clasificarea adoptată de revista Electronics — SUA, *calculatoarele personale* sînt plasate în plaja de preţuri pînă la 5000 $, iar *microcalculatoarele* (calculatoare personal-profesionale) între 5 şi 20 mii $ (incluzînd microsistemele multiutilizator şi staţiile de lucru), dar uneori în diverse lucrări se foloseşte fie denumirea de calculatoare personale, fie de microcalculatoare, în extensie pentru ambele grupe.

O scurtă prezentare privind începutul istoriei microcalculatoarelor şi calculatoarelor personale, inclusiv utilizarea lor, este dată în suplimentul revistei **„Ştiinţă şi tehnică"** din septembrie 1986, intitulat **„Calculatorul, nimic mai simplu!"**, iar o serie de articole tehnice, detaliate şi atractive, pot fi citite în revista menţionată, mai ales în numerele apărute în ultimii 3 ani.

În *ţările socialiste* s-a acordat o importanţă deosebită acestui domeniu, fiind elaborate în ultimii ani prin concepţie proprie cîteva zeci de modele de calculatoare personale, cu o fabricaţie anuală în serii care cresc rapid, de la tipurile familial şi pentru şcoală, pînă la cele profesionale avansate, ca de pildă:

- HT10802 şi PRIMO (RPU) ROBOTRON 1715 (RDG), MERITUM (RPP);
- AGAT (URSS) şi PRAVETZ (RPB). compatibile APPLE;
- ISKRA 250 (URSS), SMEP (RSC), ROBOTRON (RDG), PROPER (RPU) şi INTELEXT (RPB), primele două compatibile cu IBM PC, iar celelalte cu IBM PC XT.

Tabelul 2.3

MODEL DE CALCULATOR	ANUL LANSĂRII	MICROPROCESOR	MEMORIA INTERNĂ	PERIFERICE (OPŢIUNI),	SISTEM DE OPERARE (OPŢIUNI)	LIMBAJE/ APLICAŢII
1	2	3	4	5	6	7
ZX SPECTRUM	1982	Z80A, 8 biţi	16—48 k oct	Casetă magnetică (microdrive)	—	Interpretor de BASIC rezident şi pe casetă, Logo, Forth, Pascal, Micro-Prolog
COMMODORE 64	1982	6510, 8 biţi	64 k oct.	Casetă magnetică (disc flexibil)	Monitor (CP/M)	Interpretor de BASIC rezident, Pascal, Logo, Forth
COMMODORE 128	1984	8701 8 biţi	128—256 k oct.	Disc flexibil (caseta magnetică)	CP/M	BASIC, Fortran, Pascal, Logo, Forth, dBASE II
APPLE II C	1983	6502 8 biţi	64—128 koct.	Caseta magnetică (disc flexibil, miniimprimantă)	Apple-DOS	BASIC, Pascal, Logo, Forth, Fortran
MACINTOSH	1984	68000 16 biţi coprocesor matematic 68081	128 koct —1 Moct	Discuri flexibile, Disc fix, Miniimprimantă	Apple-DOS	BASIC, Pascal, C. Forth, Fortran, etc.
MACINTOSH II	1986	68020 32 biţi coprocesor matematic 68881/16 MHz	1—8 Moct	Discuri flexibile, Disc fix 40 Moct, Imprimantă cu laser	DOS, UNIX	Toate limbajele PC larg răspîndite şi aplicaţiile standard
IBM PC şi modelul extins IBM PC/XT	1981 1983	8086/8088 16 biţi coprocesor matematic 8087	256—640 koct.	Discuri flexibile Disc fix tip Winchester, Miniimprimantă	MS-DOS	BASIC, Fortran, Cobol, Pascal, C. Forth, Lisp.

Tabelul 2.3. (continuare)

1	2	3	4	5	6	7
IBM* PC/AT	1984	80286 16 biţi 10 MHz coprocesor matematic 80287	256 koct - -2 Moct	Discuri flexibile, Disc fix, Miniimprimantă	MS-DOS, XENIX	BASIC, Turbo-Pascal, Fortran 77, Cobol, C, Forth, Lisp, dBASE III, ORACLE, LOTUS 1—2—3, AUTO CAD
DESK PRO 386 (COMPAQ)	1987	80386 32 biţi 16/20 MHz coprocesor matematic 80387/16 MHz	1—16 Moct.	Discuri flexibile, Disc fix 100/300 Moct., Imprimantă cu laser	MS-OS/2	Toate limbajele PC larg răspîndite şi aplicaţiile standard

* Noua serie PERSONAL SYSTEM II cu modele de 16 şi 32 de biţi, lansată recent de IBM este prezentată în detaliu în capitolul 25.

Pentru ţările capitaliste dezvoltate industrial s-a ales *prezentarea structurii consumului general de tehnică de calcul direct şi indirect, într-un grup de 6 ţări, care include, SUA, Japonia, RFG, Marea Britanie, Franţa şi Italia în* 1989, *structură dată în tabelul* 2.4.

Ponderea consumului general de tehnică de calcul conform tabelului 2.4., din consumul total aferent întregii industrii de electronică din cele 6 ţări este de peste 66 %, ceea ce ilustrează în continuare atît un aport major al industriei

Tabelul 2.4

PREVIZIUNI PRIVIND CONSUMUL CUMULAT ÎN ANUL 1989, DE ECHIPAMENTE PENTRU PRELUCRAREA DATELOR, PROGRAME DE BAZĂ, APLICATIVE ŞI CÎTEVA GRUPE DE PRODUSE BAZATE PE TEHNICA DE CALCUL ŞI MICROPROCESOARE ÎNTR-UN GRUP* DE 6 ŢĂRI

— ponderea valorică, % —

TOTAL	100%
din care:	
1 — echipamente pentru prelucrarea datelor	66%
2 — programe de bază, aplicative şi instrumente pentru aplicaţii (livrate distinct)	20%
3 — echipamente pentru comunicaţii de date şi terminale facsimil	5%
4 — produse de consum personal bazate pe microprocesoare şi circuite integrate specializate	2%

5 — echipamente pentru controlul proceselor 6%

6 — sisteme CAD/CAE/CAM pentru industria electronică inclusiv microelectronică (proiectare, inginerie şi fabricaţie asistate de calculator) 1%

* Grupul de 6 ţări cuprinde: SUA, Japonia, RFG, Marea Britanie, Franţa şi Italia

NOTĂ:

— ponderile din tabelele 2.4, 2.5 şi 2.6 au fost calculate prin prelucrarea selectivă şi agregarea datelor publicate în revista Electronics şi în alte lucrări din 1988 şi 1989, considerînd consumul egal cu producţia livrată minus export plus import.

— în revista Electronics subgrupele 3, 4, 5 şi 6 sînt incluse la alte industrii din componenţa industriei electronice, dar în tabelul 2.4. au fost însumate pentru a ilustra mai bine impactul tehnicii de calcul în industrie.

Tabelul 2.5

STRUCTURA CONSUMULUI ÎN 1989 ÎN SUA, DE ECHIPAMENTE PENTRU PRELUCRAREA DATELOR ŞI PROGRAME, PE PRINCIPALELE GRUPE

— ponderea valorică, % —

Echipamente de prelucrare a datelor şi programe, total 100%

din care:

— echipamente de prelucrare a datelor, total 77%

din care:

— sisteme de calcul 54%

— echipamente periferice de intrare/ieşire, inclusiv grafice 12%

— memorii externe 7%

— terminale 4%

— programe*de bază, aplicative şi instrumente pentru aplicaţii, total 23%

din care:

— programe de sistem 12%
(sisteme de operare, instrumente de diagnosticare şi punere la punct)

— programe aplicative 6%
(prelucrare de texte, calcule tehnico-ştiinţifice, economice, etc.)

— instrumente pentru aplicaţii 5%
(baze de date, programe CAD/CAE/CAM pentru mecanică, programe de editare publicitară, de inteligenţă artificială, de prelucrare de imagini, etc.)

* Livrări distincte ale producătorilor de software (exclusiv atît programele şi microprogramele incluse în preţul sistemelor, cît şi programele proprii elaborate de utilizatori).

2. CALCULATOARE PERSONALE

Tabelul 2.6

DISTRIBUŢIA CONSUMULUI ÎN 1989 DE SISTEME DE CALCUL, PE PRINCIPALELE SUBGRUPE, COMPARATIV, ÎN SUA (I) ŞI ÎNTR-UN GRUP DE 6 ŢĂRI (II) CARE INCLUDE SUA

— ponderea valorică, % —

	I*	II*
Sisteme de calcul, total	100%	100%
din care:		
— calculatoare personale	31%	20%
— microcalculatoare multiutilizator şi staţii de lucru	12%	14%
— minicalculatoare şi superminicalculatoare	25%	21%
— calculatoare medii şi medii-mari (tip mainframes)	30%	44%
— minisupercalculatoare şi supercalculatoare	2%	1%

*I — SUA
II — SUA, Japonia, RFG, Marea Britanie, Franţa şi Italia

directe de tehnică de calcul, cît şi o creştere a ponderii grupelor de produse asociate (raportate valoric în cadrul altor industrii, dar bazate pe calculatoare şi microprocesoare).

În tabelul 2.5 se *dă structura consumului estimat de tehnică de calcul (echipamente şi programe)* în 1989 *în SUA*, ilustrînd următoarele grupe de produse cu ponderi mari:

— sisteme de calcul —54%
— memorii externe, periferice de intrare / ieşire şi terminale —23%
— programe de sistem, aplicative şi instrumente pentru aplicaţii (livrări distincte) —23%

Distribuţia consumului de sisteme de calcul pe principalele subgrupe, conform tabelului 2.6. arată că *subgrupele de calculatoare personale şi microcalculatoare, considerate împreună ajung la o pondere de 43% în acest an, devenind cel mai mare sector de tehnică de calcul din SUA, în timp ce în grupul de 6 ţări menţionate, ponderea respectivă atinge nivelul de 34%*. De altfel, începînd cu anul 1988, chiar numai ponderea staţiilor de lucru de 16 şi 32 de biţi devine mai mare decît ponderea minicalculatoarelor clasice!

În aceste condiţii, apar redistribuiri în favoarea subgrupelor cu performanţe superioare minicalculatoarelor (de fapt cu un raport performanţă/cost mai bun) şi anume mai ales spre subgrupa superminicalculatoarelor, dar şi spre calculatoarele medii, medii-mari, minisupercalculatoare şi supercalculatoare.

Cele două subgrupe apărute în ultimii ani în clasificarea adoptată pe plan mondial, pot fi definite pe scurt astfel: *superminicalculatoarele* reprezintă minicalculatoare cu performanţe de vîrf, în timp ce *minisupercalculatoarele* acoperă intervalul dintre cele mai performante supermini şi calculatoarele medii-mari sau supercalculatoare. Minisupercalculatoarele au facilităţi de prelucrare vectorială integrate, dar costă mult mai puţin decît supercalculatoarele.

O caracteristică importantă a industriei de calculatoare personale este dată de seriile mari de fabricaţie, de pildă ZX Spectrum cu peste un milion şi IBM PC cu cîteva milioane. Se estimează că la începutul anului 1988 erau instalate numai în America de Nord peste 14 milioane de calculatoare personale dintre care 10—20% sînt conectate în reţele.

Stațiile de lucru, avînd o dinamică mare a producției, reprezintă posturi ale activității inginerești de înaltă productivitate, fiind dotate cu microprocesoare de 16 și 32 de biți, afișaj grafic color, discuri magnetice rapide, imprimantă grafică, plotter și mai ales cu instrumente de programare și proiectare cu facilități grafice interactive (grafică 3D) adecvate familiei respective de aplicații în electronică, mecanică, etc.). Cu *puțin timp* în urmă, stațiile de lucru ofereau o viteză de 4—5 milioane de operații/s. (MIPS), *apoi au apărut* mai multe modele de 10 MIPS și începînd cu mijlocul anului 1988 *au fost* anunțate primele modele de 20 MIPS, fiind în elaborare modele cu viteze apreciabil mai mari.

Memoriile externe incluse în tabelul 2.5 cuprind: unități cu discuri magnetice fixe (se mai numesc tip Winchester sau rigide, suportul magnetic nefiind interschimbabil între unități), unități cu discuri magnetice flexibile, unități cu pachet de discuri amovibile, unități cu benzi magnetice încasetate și tip minicasetă, și noua subgrupă a discurilor optice.

Discurile rigide (cu suport magnetic fix cu diametrul de $3\,^1/_2$, $5\,^1/_4$, 8 sau 14 in) și discurile flexibile (cu suport magnetic flexibil cu diametrul de $3\,^1/_2$, $5\,^1/_4$ sau 8 in) au împreună o pondere valorică de aproape 84% din totalul memoriilor externe, unitățile cu disc fix avînd ponderea de 72%. Dintre aceste tipuri de discuri, cele mai des folosite în configurațiile microcalculatoarelor personale sînt unitățile cu disc fix și flexibile cu suporturi magnetice de $5\,^1/_4$ și $3\,^1/_2$ in.

Echipamentele periferice de intrare-ieșire includ: imprimante seriale, imprimante de linii de medie și mare viteză, imprimante cu laser, digitizoare, plottere, etc. Imprimantele cu impact au cea mai mare pondere valorică (38%) din echipamentele de intrare-ieșire și sînt larg răspîndite în configurațiile microcalculatoarelor, stațiilor specializate de proiectare și ale unor calculatoare personale, mai ales modele profesionale, dar și imprimantele cu laser dețin în 1989 o pondere importantă (peste 20%).

Menționăm cîteva echipamente și dispozitive periferice de mare performanță, utilizate în configurațiile calculatoarelor personale și micro, evoluate:

— **unități miniaturizate de disc Winchester de 10/20 Moct (numite Hardcard), montate pe o parte din placheta logică, deci încorporate în calculator, la fel ca orice plachetă;**
— **unități de disc optic cu laser, care permit numai citiri (utilizate de firmele IBM, DEC și APPLE) și pot avea o capacitate de 200 ÷ 600 Moct (echivalentul a 200000 pagini de text)**
— **imprimante cu laser, de pildă Laser Jet de la firma Hewlett-Packard și Laser Writer de la Apple cu viteza de tipărire de 8 pg./min. sau modelul Xerox 4045 cu 10 pg. /min. la o rezoluție maximă pentru text și grafică, de 300 × 300 puncte/in²;**
— **display grafic cu o mare putere de rezoluție, de 640 × 480 puncte adresabile pe ecran, cu selectarea a 16 culori dintr-o gamă de 256.**
— **tastatură complet autonomă cu comanda în infraroșu (introdusă prima dată la modelul IBM PC Jr.)**

Evoluția industriei de calculatoare personale și microcalculatoare ilustrează următoarele caracteristici și tendințe:

a) *„maturizarea" domeniului*, reflectată prin:

— **modificarea ritmului de creștere a vînzărilor de la 50—100% în 1980—1984, la 8—20% pentru perioada 1985—1990, ceea ce înseamnă că faza „explozivă" a fost depășită;**
— **reducerea severă a numărului de firme producătoare, concentrarea producției la firmele care pot asigura compatibilitatea deplină cu modelele devenite „standard mondial" și dezvoltă producții de mare**

serie, rentabile, inclusiv în fabrici automatizate (cele mai „agresive" produse noi au fost lansate de IBM, Apple, Compaq, Tandy şi Zenith din SUA privind gama calculatoarelor personale şi microcalculatoarelor evoluate tehnologic);

— dezvoltarea deosebită a producţiei de programe destinate aplicaţiilor de o mare diversitate, bazate pe calculatoare personale şi micro. O mare parte a elaboratorilor şi furnizorilor de programe este organizată sub forma „caselor de software", cu echipe mai mici, dar foarte productive. Modelele Sinclair Spectrum şi IBM PC beneficiază, de pildă, de cîteva mii de programe fiecare. Astfel, ponderea programelor aferente microcalculatoarelor şi calculatoarelor personale, a ajuns în anul 1988, în SUA, la peste 16% din totalul livrărilor de software;

b) *creşterea evidentă în anii 1984 —1989 a ponderii valorice* a calculatoarelor personale şi micro din totalul sistemelor de calcul (de la circa 20 la peste 40%);

c) *diversificarea programelor aplicative* pe următoarele direcţii:

— programe pentru educaţie şi învăţămînt, instruire asistată de calculator;

— programe specializate (pentru calcule tehnico-ştiinţifice, economice, financiare, de proiectare automată, conducerea proceselor de producţie);

— programe de divertisment (jocuri logice, educaţionale, etc.);

— software integrat pentru modele profesionale (incluzînd programe de grafică, baze de date, prelucrare şi actualizare de tabele, prelucrare de texte, programe pentru telecomunicaţii);

— programe pentru sisteme expert implementate pe microcalculatoare profesionale şi staţii de lucru avansate. Pachetele de programe sînt bazate pe tehnici ale domeniului AI (Artificial Intelligence), includ cunoştinţe despre sute de obiecte sau stări şi oferă soluţii respectînd seturi de cîteva sute de reguli.

d) *elaborarea unor sisteme de operare evoluate*. Sistemul de operare UNIX elaborat de firma AT & T tinde să devină începînd cu 1988/1989 un standard mondial. De altfel, în 1991 faţă de 1985/1986, ponderea valorică a sistemelor UNIX din totalul sistemelor de operare va fi de 4 ori mai mare. Privind sistemele noi de operare pentru microcalculatoare bazate pe microprocesorul Intel de 32 de biţi, 80386, este de aşteptat o competiţie între versiunea UNIX-386 şi noul sistem de operare 386-0S/2 lansat de firma Microsoft, dar şi acesta din urmă este considerat numai un produs de tranziţie pînă la apariţia unui nou standard în 1989/1990.

e) *atenuarea şi chiar dispariţia unei distincţii nete între micro, mini şi calculatoare medii*, ca urmare a introducerii microprocesoarelor de 32 de biţi. Această afirmaţie trebuie înţeleasă în sensul că microcalculatorul de azi, înlocuieşte prin performanţă minicalculatorul de ieri. Dar toate grupele actuale din cadrul sistemelor de calcul au performanţe noi şi sînt adresate unor sfere noi de aplicaţii. De pildă, în SUA, valoarea totală anuală a instalărilor de supercalculatoare s-a dublat în ultimii trei ani, ritm extrem de rapid. De altfel, într-o perioadă de 13 ani pînă la începutul anului 1985, au fost instalate primele 100 de supercalculatoare Cray, iar în numai doi ani şi jumătate pînă în septembrie 1987, se ajunsese la cel de-al 200-lea sistem livrat. Experţii au estimat că în următorii 10 ani vor fi instalate în lume cîteva mii de supercalculatoare dedicate aplicaţiilor ştiinţifice şi inginereşti, cu puteri de calcul de minim 100—200 milioane de operaţii în virgulă mobilă/secundă şi la preţuri de 1—50 milioane $ (printre cele mai rapide supercalculatoare, introduse recent, fiind Cray-3 şi ETA GF 10 cu viteze de 10 miliarde de operaţii în virgulă mobilă/secundă). Supercalculatoarele au devenit un standard şi nu o excepţie,

fiind necesare pentru o serie de aplicaţii cu calcule laborioase privind: meteorologia, mecanica fluidelor, fizica plasmei, fizica atomică şi nucleară, ingineria nucleară, sistemele ecologice, prelucrarea digitală a imaginilor recepţionate de la sateliţi, prelucrarea grafică şi recunoaşterea formelor, inventarierea resurselor terestre, modele globale ale sistemelor socio-economice, dar şi pentru simulări în chimia computaţională, în industriile de electronică, auto şi aerospaţială, respectiv în industria petrolieră.

Astfel, *dacă în urmă cu 5 ani existau numai 10 programe aplicative pentru supermaşinile Cray, astăzi există peste 450 de programe aplicative*, iar la unele modele, instalate în universităţi, se pot conecta prin staţii de lucru sau calculatoare personale pînă la 5000 de utilizatori per sistem.

Dezvoltarea domeniului tehnicii de calcul este atît de rapidă, încît începînd cu anul 1988, dar mai ales din 1989, are loc o înlocuire treptată atît a superminicalculatoarelor cît şi a calculatoarelor medii-mari clasice, cu 3 noi tipuri de calculatoare: cu arhitecturi paralele, specializate pentru prelucrarea on-line a tranzacţiilor şi cu set redus de instrucţiuni (denumite în literatură RISC, de la Reduced — Instruction — Set — Computers).

După unii experţi, introducerea cu succes a unor noi sisteme este condiţionată în 1989 de îndeplinirea a cel puţin 4 criterii principale şi anume:

— raport performanţă / preţ optim;
— facilităţi de interconectare;
— sistem de operare UNIX standard;
— interfeţe prietenoase cu utilizatorul.

2.4. Industria românească de calculatoare personale şi personal-profesionale

După anul 1976 şi mai ales în ultimii 3÷5 ani, s-a pus un accent deosebit pe elaborarea şi asimilarea unor modele noi de calculatoare personale şi personal-profesionale sau microcalculatoare. Cîteva modele produse în serie, au fost incluse în enumerarea din tabelul 1.3.

În cele ce urmează, vom da o clasificare pentru calculatoarele personale şi personal-profesionale româneşti (aflate în stadii diferite, de la prototip la producţie de serie şi prezentate de la simplu la complex, inclusiv modelele care nu se mai fabrică):

1 — calculatoarele personale de 8 biţi, cu o concepţie proprie, fără compatibilitate cu modelele larg răspîndite pe plan mondial, avînd casetofon şi TV alb/negru, respectiv interpretor de BASIC (aMIC şi PRAE, fabricate la FMECTC — Timişoara pe baza modelelor elaborate de IPB şi ITCI Cluj-Napoca cu menţiunea că pe modelul aMIC s-a implementat şi limbajul FORTH);

2 — calculatoare personale de 8 biţi, compatabile cu modelul SPECTRUM, cu casetofon şi TV sau monitor alb-negru/color, (HC-85 asimilat în producţie de serie la ICE Bucureşti pe baza modelului elaborat de IPB, TIM-S asimilat la FMECTC Timişoara în colaborare cu IPT, COBRA elaborat de Filiala din

---

* O serie de elemente suplimentare despre supercalculatoare se află în grupajul de articole publicat în revista ŞTIINŢĂ şi TEHNICĂ Nr. 4 din 1987 şi în cîteva numere apărute în 1988 şi 1989.

Brașov a ITCI și fabricat în regim de microproducție, HC-85 avînd și o versiune cu disc flexibil);

3 — calculatoarele personale de 8 biți, cu selectarea compatibilității din două variante, cu casetofon și/sau unitate de disc flexibil, TV sau monitor alb-negru/color (COBRA cu selectarea compatibilității SPECTRUM sau CP/M, aflat în regim de microproducție pe baza modelului elaborat de Filiala Brașov a ITCI, respectiv noul model HC-88);

4 — calculatoare personal-profesionale de 8 biți cu memorie internă de 64 koct, compatibile CP/M, cu monitor alb-negru, unități de discuri flexibile (M-118 și CUB-Z la ICE București, TPD și Junior la IEPER București și CE 119 sau L/B 881 la ITCI-microproducție);

5 — calculatoare personal-profesionale de 16 biți, compatabile cu sistemul de operare MS-DOS sau de tip UNIX-micro (sistemul românesc de operare „U"), cu memorie internă de minim 128/256 Koct. și maxim 640/1024 Koct. cu monitor alb-negru/color, unități de discuri flexibile, disc Winchester, imprimantă matricială și plotter (M-216 cu capacitatea maximă de 1024 Koct. și FELIX-PC* cu capacitatea maximă de 640 Koct. și compatibilitate cu modelul IBM PC, ambele asimilate la ICE București pe baza modelelor elaborate de IPB, respectiv modelul Junior XT compatibil cu IBM PC/XT, în curs de asimilare la IEPER)

Se evidențiază astfel în țara noastră, *trei direcții și compatibilități* în domeniul calculatoarelor personale și personal-profesional, bine definite: și anume:

1 — Sinclair Spectrum-BASIC

2 — sistem de operare CP/M pentru calculatoare de 8 biți

3 — sistem de operare MS-DOS sau tip UNIX-micro, pentru calculatoare de 16 biți tip IBM-PC și PC/XT.

Ținînd seama de aceste compatibilități, cel mai mare număr de calculatoare personale produse în 1985—1989 este din clasa compatabil SPECTRUM, modelul HC-85 avînd seria de fabricație cea mai mare.

*** În cadrul părții a XI-a se prezintă distinct caracteristicile generale ale calculatorului FELIX-PC.**

Partea a II-a.

CALCULATOARE NUMERICE. REALIZARE FIZICĂ —BAZELE ARITMETICE ŞI LOGICE.

Capitolul 3 Baze aritmetice.

Calculatoarele electronice sînt conțruite pe baza unor dispozitive fizice (tranzistoare, circuite integrate logice etc.), care funcţionează ca elemente cu două stări (conducţie/blocat), cărora le sînt asociate două niveluri de tensiune (coborît/ridicat). Informaţia elementară manipulată de calculator va fi astfel asociată cu cele două niveluri de tensiune: coborît (0V) şi ridicat (+5V), care vor fi folosite pentru reprezentarea numerelor cu ajutorul unui limbaj binar, al cărui alfabet A constă din două caractere:

$$\mathbf{A}=\{0,1\}$$

În acest limbaj cuvintele vor fi de forma: 0, 1, 10, 11, 100, 101, 110, 111 etc. Ele pot fi tratate ca numere binare sau drept coduri ale elementelor unei mulţimi:

$$0 \leftarrow\rightarrow a_0$$
$$1 \leftarrow\rightarrow a_1$$
$$10 \leftarrow\rightarrow a_2$$

3.1. Sistemul de numeraţie binar

Elementele limbajului binar pot fi folosite pentru reprezentarea numerelor în sistemul de numeraţie în baza doi. În acest scop fiecare element al cuvîntului va reprezenta coeficientul unei puteri a lui doi, elementul din extrema dreaptă

fiind coeficientul lui doi la puterea 0. Ponderile coeficienţilor (puterile lui doi) cresc de la dreapta, la stînga. Acesta este un sistem de numeraţie poziţional.

În cazul general un număr N va putea fi scris în baza q astfel

$$N=a_{n-1}\ q^{n-1}+a_{n-2}\ q^{n-2}+\ldots\ldots+a_iq^i+\ldots\ldots+a_1q^1+a_0q^0$$

unde: $0\leqslant a_i<q$.

sau prescurtat:

$$N_{(q)}=a_{n-1}a_{n-2}\ldots a_i\ldots a_1a_0$$

Se consideră cazurile particulare, cu bazele: $q=2$ şi $q=10$.

Pentru $q=10$ avem:

$$N=b_{n-1}\ 10^{n-1}+\ldots+b_i10^i+\ldots+b_010^0$$

unde: $0\leqslant b_i<10$,

sau prescurtat:

$$N_{(10)}=b_{n-1}b_{n-2}\ldots b_i\ldots b_0$$

Pentru $q=2$ avem:

$$N=c_{p-1}2^{p-1}+\ldots+c_i2^i+\ldots+c_02^0$$

unde: $0\leqslant c_i<2$,

sau prescurtat:

$$N_{(2)}=c_{p-1}c_{p-2}\ldots c_i\ldots c_0$$

În sistemul de numeraţie binar, o cifră binară mai poartă numele de bit (*rang binar — binary digit* — în engleză).

Ca valoare numărul zecimal 1987 este interpretat astfel:

$$1\cdot 10^3+9\cdot 10^2+8\cdot 10^1+7\cdot 10^0=1000+900+80+7.$$

Numărul binar 1110 va fi interpretat ca valoare astfel:

$$1\cdot 2^3+1\cdot 2^2+1\cdot 2^1+0\cdot 2^0=8+4+2+0=14$$

El va reprezenta în binar numărul întreg 14, în baza 10.

Un număr binar cu n biţi permite reprezentarea unor numere întregi în gama: $0\leqslant N\leqslant 2^n-1$.

EXEMPLE:

— un cuvînt de 8 biţi permite reprezentarea întregilor între 0 şi $2^8-1=255$.

— un cuvînt de 16 biţi permite reprezentarea întregilor între 0 şi $2^{16}-1=65535$.

Numărul de biţi n, necesari într-un cuvînt binar, pentru a reprezenta un număr întreg N, dat în baza 10, se poate calcula cu ajutorul formulei:

$$n=\lceil log_2N \rceil \text{ biţi.}$$

unde simbolul $\lceil\ \rceil$ reprezintă valoarea întreagă imediat superioară lui log_2N.

Practic, n se poate calcula prin încadrarea lui N între puterile lui doi.

$$2^{n-1} < N \leqslant 2^n$$

Dacă $N=1987$, rezultă:

$$N = \lceil log_2 N \rceil = 11$$

sau:

$$2^{10} = 1024 < 1987 < 2^{11} = 2048$$

3.2. Conversia binar – zecimală

Conversia binar-zecimală se poate realiza pe baza următoarei secvenţe de operaţii (algoritm):

1. Se execută o poziţionare la dreapta numărului.
2. Se iniţiază: $N=0$ şi $n=0$.
3. Se citeşte următorul caracter din stînga.
4. Dacă caracterul este un spaţiu liber (blanc), atunci STOP.
5. Dacă caracterul este zero, atunci se incrementează n cu o unitate $(n+1)$ şi se trece la 3; dacă caracterul este unu, atunci se calculează $N+2^n$, se incrementează n cu o unitate $(n+1)$ şi se trece la 3.

EXEMPLU

100111 reprezintă, în zecimal, numărul: $N=1+2+4+32=39$

Numerele binare terminate în zero sînt numere pare, în timp ce numerele binare terminate în 1 sînt numere impare.

3.3. Conversia zecimal – binară.

Această conversie se realizează prin împărţiri succesive ale cîtului, de la împărţirea precedentă, la noua bază — doi, şi reţinerea restului curent, pînă cînd cîtul curent devine mai mic decît noua bază. Ca prim cît se ia numărul N. Resturile obţinute reprezintă cifrele numărului binar, în ordinea crescătoare a ponderilor. Cifra cu ponderea cea mai mare este primul cît obţinut, cu valoarea mai mică decît doi.

Fie numărul întreg N, în baza 10. El poate fi convertit în binar astfel:

$$N = 2.q_0 + r_0, \qquad 0 \leqslant r_0 \leqslant 1.$$

Dacă: $q_0 \geqslant 2$, atunci:

$$q_0 = 2 \cdot q_1 + r_1, \qquad 0 \leqslant r_1 \leqslant 1.$$

Dacă: $q_1 \geqslant 2$, atunci:

$$q_1=2 \cdot q_2+r_2, \qquad 0 \leqslant r_2 \leqslant 1.$$

Dacă: $q_i<2$, atunci:

$$N=q_i \cdot 2^i+...+r_2 \cdot 2^2+r_1 \cdot 2^1+r_0$$

Fie numărul $N=37$. Operaţiile implicate de conversie vor fi următoarele:

$$\begin{array}{llllll}
 & 37 \mid 2 & & & & \\
r_0= & 1 & 18 \mid 2 & & & \\
 & r_1= & 0 & 9 \mid 2 & & \\
 & & r_2= & 1 & 4 \mid 2 & \\
 & & & r_3= & 0 & 2 \mid 2 \\
 & & & & r_4= & 0 \quad q_0=1
\end{array}$$

rezulta:

$$N_2=100101$$

3.4. Reprezentările octală şi hexazecimală

Numerele binare sînt practic greu de manipulat, avînd prea multe ranguri. În scopul scrierii lor sub o formă condensată, se folosesc sistemele de reprezentare octal şi hexazecimal. Avînd ca baze puteri ale lui doi (8 şi respectiv 16), ele permit o conversie mecanică, fără calcule, între reprezentările octală/hexazecimală şi respectiv binare.

Sistemul octal foloseşte un alfabet A cu 8 cifre:

$$A=\{0, 1, 2, 3, 4, 5, 6, 7\}$$

Conversia binar-octală este următoarea:

binar	octal	binar	octal
000	0	100	4
001	1	101	5
010	2	110	6
011	3	111	7

Numărul 110101 se scrie în octal sub forma 65 deoarece: ,

$$110=6_{(8)}, \text{ iar } 101=5_{(8)}.$$

Conversia binară a unui număr octal rezultă prin înlocuirea fiecărei cifre octale cu triada binară echivalentă.

Sistemul hexazecimal foloseşte un alfabet A, cu 16 cifre:

$$A=\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$$

Mai jos se dau unele echivalenţe între numere zecimale, hexazecimale şi binare:

ZECIMAL	HEXAZECIMAL	BINAR
0	0	0000
1	1	0001
2	2	0010
...	...	...
10	*A*	1010
11	*B*	1011
12	*C*	1100
13	*D*	1101
14	*E*	1110
15	*F*	1111
16	10	0001 0000
17	11	0001 0001

Conversia unui număr binar în hexazecimal se realizează prin împărţirea numărului binar în tetrade, începînd de la dreapta spre stînga, şi înlocuirea lor cu cifrele hexazecimale corespunzătoare:

$$10001101 = 8D_{(16)}$$

3.5. Codul binar–zecimal (2–10)

Cifrele sistemului de numeraţie zecimal pot fi înlocuite cu tetrade binare, rezultînd astfel posibilitatea de a reprezenta direct în calculatoare cifrele zecimale, fără a fi necesară o conversie prealabilă.

Astfel, cifrele zecimale au următarele tetrade binare echivalente:

cifra zecimală	tetrada binară
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Tetradele 1010, 1011, 1100, 1101, 1110, 1111 sînt incorecte, neavînd cifre zecimale asociate.

EXEMPLE:

65 =01100101
88 =10001000

Conversia „binar — zecimal“ — „zecimal“ se realizează mecanic.

3.6. Adunarea şi înmulţirea numerelor binare

Adunarea are loc pe baza regulilor concretizate în următorul tabel:

$$0+0=0$$
$$0+1=1$$
$$1+0=1$$
$$1+1=10$$

În ultima linie se constată apariţia transportului egal cu unu.

EXEMPLU:

```
 1 1 1   1
 ↖ ↖↖   ↖
  1 1 1 0 1          29
+ 1 0 1 0 1         +21
-----------         ---
1 1 0 0 1 0          50
```

În partea superioară au fost marcaţi biţii de transport la adunarea fiecărui rang.

Dacă numărul de biţi, din cuvintele binare, este fixat la o valoare dată n (5 în cazul de mai sus), la adunare poate apărea pericolul depăşirii capacităţii de reprezentare, cînd suma obţinută este mai mare decît 2^5-1 (31 — în exemplul de mai sus).

Înmulţirea numerelor binare se bazează pe regulile din tabelul de mai jos:

$$0\times0=0$$
$$0\times1=0$$
$$1\times0=0$$
$$1\times1=1$$

Algoritmul înmulţirii a două numere binare este următorul:

1. Se iniţializează $N=0$
2. Se iniţializează rezultatul (produsul) $P=0$.
3. Se examinează cifrele înmulţitorului începînd de la dreapta spre stînga.
4. Dacă cifra curentă este absentă (blanc), atunci STOP.
5. Dacă cifra curentă este 0, se incrementează N cu o unitate ($N+1$) şi se trece la 3.
6. Dacă cifra curentă este 1, deînmulţitul, deplasat cu N ranguri spre stînga, se adună la P; se incrementează N cu o unitate şi se trece la 3.

EXEMPLU:

```
     1101        13 ×
 ×    101         5
 --------       ----
     1101        65
   1101
 --------
  1000001
```

Înmulţirea a două numere binare de cîte n biţi generează un rezultat cu $2n$ biţi.

3.7. Reprezentarea numerelor negative

3.7.1. Reprezentarea numerelor în modul şi semn.

În scopul reprezentării semnului unui număr binar, se va aloca bitul plasat în extrema stîngă a numărului dat.

Astfel, dacă pentru reprezentarea numărului se folosesc *n* biţi, bitul de rang (*n*—1) va fi folosit pentru codificarea semnului conform convenţiei de mai jos:

„+"<—>„0"

„—"<—>„1"

Fie numerele +6 şi —6, reprezentate în binar, cu cîte 8 ranguri, inclusiv rangul de semn:

00000110 reprezintă +6

semn—↕

10000110 reprezintă —6

Acest mod de reprezentare are o serie de dezavantaje:

— biţii de semn ai numerelor care participă la operaţia aritmetică dată, trebuie trataţi separat,

— apare o configuraţie de număr binar, 1000...00, care trebuie interpretată ca zero (zero negativ), existînd astfel două reprezentări pentru zero.

— se impune definirea unei operaţii de scădere a valorilor absolute ale numerelor.

3.7.2. Reprezentarea în complementul faţă de 1.

Numărul negativ în complementul faţă de 1 se va obţine prin înlocuirea fiecărei cifre binare a numărului dat, prin complementul ei faţă de 1.

Astfel, complementul faţă de 1, al lui 0, este 1, iar complementul faţă de 1, al lui 1, este 0.

Un număr pozitiv are în complementul faţă de 1 aceeaşi reprezentare ca în modul şi semn.

EXEMPLU:

$+5_{(10)} \leftarrow\rightarrow 0101$

$-5_{(10)} \leftarrow\rightarrow 1010$

Se constată că bitul cel mai semnificativ este folosit pentru reprezentarea semnului.

Adunarea bit cu bit a unui număr binar şi a complementului său faţă de 1, generează un rezultat egal cu zero, reprezentat printr-un număr ai cărui biţi sînt egali cu 1.

Astfel, la adunarea lui +5 cu —5 se obţine: 1111, ceea ce reprezintă complementul faţă de 1 al lui 0000.

În această reprezentare există două forme pentru zero (+0 şi —0), ceea ce constituie un inconvenient în realizarea circuitelor electronice pentru operaţiile de adunare.

Se observă că operaţia de scădere se reduce la o operaţie de adunare la descăzut a complementului faţă de 1 al scăzătorului.

La adunarea numerelor reprezentate în complementul faţă de 1, transportul care apare la stînga rangului de semn se va aduna ciclic la ultimul bit din dreapta, al rezultatului. Rangurile de semn ale numerelor participă şi ele la operaţia de adunare.

EXEMPLU:

```
  (+6)       0110
 +(−5)      +1010
 -----      -----
   1        10000
            |  +1
            -----
             0001
```

Pentru a evita neajunsurile acestei reprezentări s-a recurs la folosirea complementului faţă de 2.

3.7.3. Reprezentarea numerelor în complementul faţă de 2.

Se consideră operaţia de scădere a cifrelor zecimale 8—6=2 Dacă, în loc de —6, la 8 se aduna complementul faţă de 10 al lui 6 (adică 4) se va obţine: 8+4=12. Rezultatul (2) va fi corect, dacă se va neglija transportul (1).

În mod similar, în binar, numerele negative vor fi înlocuite prin complementul faţă de 2. Acesta se obţine (pentru numerele negative) generînd complementul faţă de 1 şi adunînd o unitate în rangul cel mai puţin semnificativ al rezultatului.

Fie numărul binar: —0001001.
Complementul faţă de 1 va fi: 11110110
Complementul faţă de 2: 11110111 se va obţine prin adunarea unei unităţi în rangul cel mai puţin semnificativ al complementului faţă de 1, al numărului dat (11110110).

Într-un cod binar cu patru cifre, reprezentările complementelor faţă de 2, ale numerelor +6 şi —6 vor fi:

+6 0110

—6 1010

Dacă se efectuează adunarea, se obţine:

```
  (+6)       0110
 +(−6)      +1010
 -----      -----
   0        10000
            ↑
            se neglijează.
```

Se constată că zero are o singură reprezentare în complementul faţă de 2.

Adunarea codurilor complementare a două numere se face rang cu rang, inclusiv rangurile de semn, neglijîndu-se transportul în afara rangului de semn.

Mai jos se dau numerele binare cu patru biţi, care codifică în complementul faţă de doi, numerele întregi cuprinse între —8 şi +7.

1000	−8	0111	7
1001	−7	0110	6
1010	−6	0101	5
1100	−4	0100	4
1011	−5	0011	3
1101	−3	0010	2
1110	−2	0001	1
1111	−1	0000	0

Prin definiție se consideră 1000 reprezentarea binară, în complementul față de 2, a numărului 8.

Complementul față de doi este utilizat în reprezentarea numerelor, în toate calculatoarele moderne.

Cu ajutorul a n biți se pot reprezenta, în complementul față de doi, numerele N, cuprinse în gama:

$$-2^{n-1} \leqslant N \leqslant 2^{n-1}-1$$

EXEMPLE:

pentru n = 8, rezultă: $-2^7 \leq N \leq 2^7-1$ sau $-128 \leq N \leq 127$.

pentru n = 16, rezulta: $-2^{15} \leq N \leq 2^{25}-1$ sau $-32768 \leq N \leq 32767$.

3.8. Reprezentarea numerelor reale

În prelucrarea numerelor reale se impune definirea unor modalități de reprezentare a părților întreagă și subunitară, precum și a virgulei.

În practică se întîlnesc modurile de reprezentare în virgulă fixă și în virgulă mobilă.

3.8.1. Reprezentarea în virgulă fixă.

Reprezentarea în virgulă fixă presupune, în cazul general, existența unei părți întregi și a uneia subunitare, despărțite printr-o virgulă a cărei poziție este fixă. De asemenea, mai sînt fixați numărul de biți afectați fiecărei părți.

Dacă se consideră un cuvînt de opt biți, din care primii patru sînt folosiți de partea întreagă și ultimii patru, de partea subunitară, vom avea:

$$N=a_3 \cdot 2^3+a_2 \cdot 2^2+a^1 \cdot 2^1+a_0 \cdot 2^0+a_{-1} \cdot 2^{-1}+a_{-2} \cdot 2^{-2}+a_{-3} \cdot 2^{-3}+a_{-4} \cdot 2^{-4}$$

În cazul în care N>0, gama de reprezentare va fi:

$$0000{,}0000 \leqslant N \leqslant 1111{,}1111 \quad \text{sau}$$
$$0{,}0 \leqslant N \leqslant 15{,}9375.$$

Dacă N ia valori, atît pozitive, cît și negative, folosind bitul cel mai semnificativ, pentru codificarea semnului, gama de reprezentare va fi:

$$-8{,}9375 \leqslant N \leqslant 7{,}9375.$$

Se constată astfel, pentru un număr dat de ranguri binare, o gamă relativ restrînsă în care se pot reprezenta numerele reale. Numărul de ranguri este limitat din considerente economice și constructive.

În calculatoarele moderne poziția virgulei este plasată la dreapta numărului, operîndu-se numai cu întregi, în formatul cu virgula fixă. Tratarea numerelor subunitare necesită introducerea unor factori de scară în programul care se execută pe calculator.

3.8.2. Reprezentarea în virgulă mobilă.

În acest caz, un număr fracționar dat, de exemplu:

$$143{,}73675 \times 10^2$$

se va reprezenta sub forma unei mantise înmulțită cu 10 la un anumit exponent. Pentru exemplul dat exista mai multe forme echivalente:

$$143{,}73675 \times 10^2$$

$$0{,}14373675 \times 10^5$$

$$0{,}014373675 \times 10^6$$

$$14373675 \quad \times 10^{-3}$$

În funcție de valoarea exponentului, se deplasează și poziția virgulei — de aici și denumirea de virgulă mobilă.

Pentru a înlătura eventualele ambiguități se va considera întotdeauna mantisia subunitară și normalizată la dreapta, adică cifra plasată imediat la dreapta virgulei va fi diferită de zero.

În exemplul de mai sus forma corectă, normalizată va fi:

$$0.14373675 \times 10^5$$

Astfel, în reprezentarea în virgulă mobilă, numărul va fi caracterizat prin mantisa, afectată de semn, și prin exponent, de asemenea, afectat de semn.

În binar, în mod similar, numărul se va scrie sub forma unei mantise normalizată, înmulțită cu doi, la un exponent corespunzător.

De exemplu numărul:

10111.1011

0.101111011 ×10 ↑ 0101* 2 ↑ 5

unde mantisa este: 0.101111011, iar exponentul are valoarea: 0101.

Atît mantisa, cît și exponentul sînt pozitive, fapt indicat prin valoarea „0" plasată în rangurile de semn.

În concluzie, un număr în virgula mobilă este caracterizat printr-un număr binar, pozitiv sau negativ, subunitar (cu virgula plasată la dreapta rangului de semn), în valoare absolută $\geqslant 0{,}5$, *numit mantisă, și printr-un număr binar întreg, pozitiv sau negativ, reprezentînd exponentul.*

---

* Simbolul ↑ semnifică ridicarea la putere iar, virgula zecimală va fi reprezentată prin punct.

De exemplu, într-o implementare dată (biblioteca aritmetică în virgulă mobilă pentru microcalculatorul HC-85), pentru exponent pot fi alocaţi 8 biţi (un octet sau byte), în care este inclus şi semnul, iar pentru mantisa 32 de biti (patru octeţi), inclusiv semnul. Aceasta va permite manipularea unor numere cu o precizie de 9—10 cifre zecimale, într-o gamă de valori cuprinse între 10 ↑ 38 şi 4.10 ↑ —39.

3.9. Reprezentarea caracterelor alfanumerice

Calculatoarele prelucrează, atît informaţie numerică, cît şi texte. Aceasta impune folosirea unor coduri binare pentru reprezentarea cifrelor, literelor şi a altor informaţii cu caracter de comandă, semne aritmetice, de punctuaţie etc.

Pentru codificarea literelor (mari şi mici) şi a cifrelor, se poate folosi un cod binar de 7 biţi (2↑7 =128 semne diferite).

Din motive legate de creşterea fiabilităţii s-a introdus şi cel de-al optulea bit, numit bit de paritate. Acesta se adaugă automat la codul de 7 biţi, al caracterului, pentru a face ca numărul total de unităţi, din codul binar respectiv, să fie par sau impar. Odată convenţia aleasă, se pot face verificările de paritate pe parcurs, pentru a constata o eventuală eroare în transmisia datelor.

În practică, pentru caractere, se folosesc două tipuri de coduri ASCII şi EBCDIC.

Mai jos se prezintă codul ASCII

Caracter sau Comanda	ASCII (hexa)	Caractere speciale	ASCII (hexa)	Caractere majuscule	ASCII (hexa	Caractere mici	ASCII (hexa)
NUL	00	(SP) spaţiu	20	@	40		60
SOH	01	!	21	A	41	a	61
STX	02	"	22	B	42	b	62
ETX	03	#	23	C	43	c	63
EOT	04	$	24	D	44	d	64
ENQ	05	%	25	E	45	e	65
ACK	06	&	26	F	46	f	66
BEL	07	'	27	G	47	g	67
BS	08	(	28	H	48	h	68
HT	09	)	29	I	49	i	69
LF	0A	*	2A	J	4A	j	6A
VT	0B	+	2B	K	4B	k	6B
FF	0C	,	2C	L	4C	l	6C
CR	0D	—	2D	M	4D	m	6D
SO	0E	.	2E	N	4E	n	6E
SI	0F	/	2F	O	4F	o	6F

DLE	10	0	30	P	50	p	70
DC1	11	1	31	Q	51	q	71
DC2	12	2	32	R	52	r	72
DC3	13	3	33	S	53	s	73
DC4	14	4	34	T	54	t	74
NAK	15	5	35	U	55	u	75
SYN	16	6	36	V	56	v	76
ETB	17	7	37	W	57	w	77
CAN	18	8	38	X	58	x	78
EM	19	9	39	Y	59	y	79
SUB	1A	:	3A	Z	5A	z	7A
ESC	1B	;	3B	[	5B	{	7B
FS	1C	<	3C	\	5C	¦	7C
GS	1D	=	3D	]	5D	}	7D
RS	1E	>	3E	∧	5E	~	7E
US	1F	?	3F	—	5F	DEL	7F

Capitolul 4

Logica matematică și circuitele logice

4.1. Generalități

În sensul cel mai larg, logica este știința care studiază formele și legile gîndirii, iar logica matematică este știința care utilizează metodele matematice pentru soluționarea problemelor de logică.

Domeniul logicii matematice, întitulat „algebra logicii", a găsit o largă aplicabilitate în analiza și sinteza schemelor care intră în componența calculatoarelor numerice.

Algebra logicii operează cu aserțiuni, reprezentînd afirmații despre care se poate spune că sînt fie adevărate, fie false.

Aserțiunile de tipul: „BASIC este un limbaj de programare conversațional", „sistemul de numerație binar folosește două simboluri pentru reprezentarea cifrelor", „la nivelul mării apa fierbe la 90°C" etc., sînt evaluate numai prin prisma adevărului sau falsității lor, independent de conținut. Unei aserțiuni adevărate i se atribuie valoarea 1, iar unei aserțiuni false — valoarea 0. O aserțiune nu poate fi în același timp falsă și adevărată. Două aserțiuni sînt echivalente, din punctul de vedere al algebrei logicii, dacă sînt simultan adevărate sau false.

Aserțiunile pot fi simple și compuse, cele din urmă obținîndu-se ca rezultat al reunirii lor cu ajutorul unor legături logice de tipul conjuncției, disjuncției, negației etc.

Pentru simplificarea calculelor cu aserțiuni, algebra logicii folosește metoda simbolică, înlocuind disjuncția, conjuncția și negația cu simbolurile: „$\cup$", „$\cap$" și „$-$", iar aserțiunile simple și compuse cu simboluri literale — litere.

În cadrul algebrei logicii aserțiunile simple și cele compuse iau valori din mulțimea bivalentă [0, 1].

Aserțiunile compuse mai poartă numele și de funcții logice. Ele pot fi reprezentate prin tabele de adevăr, care conțin valorile variabilelor (aserțiunile simple), ca argumente, și valorile corespunzătoare ale funcției. În general funcțiile logice se mai pot reprezenta cu ajutorul „formelor canonice", diagramelor Karnaugh, Veitch etc.

Adevărul sau falsitatea unei aserțiuni compuse sînt determinate de valoarea aserțiunilor simple, din care se compune, precum și de natura legăturilor logice.

Legătura logică numită disjuncție se poate pune în evidență în structura unei aserțiuni compuse, notată cu y, ca funcție de două aserțiuni simple, notate cu x_0 și x_1:

$$y \text{ este } x_0 \textit{ sau } x_1,$$

ceea ce se mai poate scrie sub formă simbolică astfel:

$$y = x_0 \cup x_1$$

În mod similar, conjuncția și negația se pot pune în evidență prin aserțiuni compuse de tipul:

$$y \text{ este } x_0 \textit{ și } x_1, \text{ adică:}$$

$$y = x_0 \cap x_1,$$

respectiv:

$$y \text{ este } \textit{non } x, \text{ sau}$$

$$y = \bar{x}$$

În tabela 4.1 sînt date, sub forma de tabele de adevăr, funcțiile logice: disjuncția, conjuncția și negația.

Tabela 4.1

TABELELE DE ADEVĂR PENTRU: DISJUNCȚIE, CONJUNCȚIE ȘI NEGAȚIE

Disjuncția

x_1	x_0	y
0	0	0
0	1	1
1	1	1
1	1	1

Conjuncția

x_1	x_0	y
0	0	0
0	1	0
1	0	0
1	1	1

Negația

x	y
0	1
1	0

Datorită asemănării între operațiile aritmetice de adunare și înmulțire, pe de-o parte, și operațiile (legăturile) logice, disjuncție și conjuncție, pe de altă parte, acestea din urmă au mai primit denumirile de: sumă logică și respectiv — produs logic.

Dacă două funcții logice, de aceleași variabile, iau valori identice, pentru aceleași seturi de variabile, ele sînt identice.

De exemplu:

$$\overline{\overline{(y)}} = y$$

reprezintă identitatea între funcția logică inițială și dubla ei negație.

Suma logică și produsul logic se bucură de proprietățile de:

comutativitate:

$$x_1 \cup x_0 = x_0 \cup x_1$$

$$x_1 \cap x_0 = x \cap_0 x_1$$

asociativitate:

$$(x_2 \cup x_1) \cup x_0 = x_2 \cup (x_1 \cup x_0)$$

$$(x_2 \cap x_1) \cap x_0 = x_2 \cap (x_1 \cap x_0)$$

distributivitate:

$$x_2 \cup (x_1 \cap x_0) = (x_2 \cup x_1) \cap (x_2 \cup x_0)$$

$$x_2 \cap (x_1 \cup x_0) = (x_2 \cap x_1) \cup (x_2 \cap x_0)$$

4.2. Formele canonice ale funcţiilor logice

Funcţiile logice se pot reprezenta, fie prin tabele de adevăr, fie prin aşa-numitele forme canonice. O funcţie logică de mai multe variabile trebuie să fie definită pentru toate combinaţiile posibile ale variabilelor.

După cum se poate constata uşor, o funcţie logică de n variabile logice trebuie să fie definită pentru toate cele 2^n combinaţii (seturi de valori) ale variabilelor. Astfel, tabela de adevăr va avea $n+1$ coloane şi 2^n linii.

În tabela 4.1 disjuncţia şi conjuncţia, ca funcţii de două variabile, sînt definite pentru $2^2=4$ combinaţii posibile ale variabilelor, în timp ce negaţia, ca funcţie de o singură variabilă este definită pentru $2^1=2$ valori ale acesteia.

Manipularea funcţiilor logice cu ajutorul tabelelor de adevăr devine greoaie, mai ales în scopul efectuării unor simplificări.

Formele canonice oferă, în această privinţă, numeroase avantaje.

Examinînd tabela de adevăr pentru conjuncţie, se constată că funcţia y_2 ia valoarea 1 numai cînd x_1 şi x_2 sînt simultan 1, ceea ce se mai poate scrie şi sub forma de produs:

$$y_2 = x_1 \cap x_2{}^{*)}$$

Plecînd de la tabela de adevăr a disjuncţiei, funcţia corespunzătoare y_1 se mai poate scrie şi sub forma unei disjuncţii de produse logice:

$$y_1 = \bar{x}_1 x_0 \cup x_1 \bar{x}_0 \cup x_1 x_0$$

Astfel, o tabelă de adevăr, pentru o funcţie de n variabile, poate fi interpretată ca o sumă logică a produselor logice P_i (în care intervin n variabile diferite, unele eventual negate), pentru care funcţia y ia valoarea 1:

$$y = \bigcup_{i=0}^{2^n-1} a_i \cdot P_i$$

unde:

— produsul logic P_i, de forma $x_0 \bar{x}_1 \ldots x_i \bar{x}_{i+1} \ldots x_{n-1}$, poartă numele de **minterm** sau **constituent al unităţii**. Indicele i are valoarea zecimală a numărului binar cu n ranguri, obţinut prin înlocuirea variabilelor negate cu 0 şi a celor fără negaţie cu 1, în expresia lui P_i. Mintermul P_i ia valoarea 1 numai pentru produsul logic i al variabilelor;

---

* În continuare, pentru conjuncţie nu se va mai folosi simbolul "∩". Expresia constituită din două sau mai multe variabile scrise alăturat va semnifica produsul logic al acelor variabile.

— coeficientul a_i ia valorile 0 sau 1, după cum funcţia ia valorile 0 sau 1, pentru setul i al variabilelor.

Pentru disjuncţie se poate scrie:

$$y_1 = 0 \cdot P_0 \cup 1 \cdot P_1 \cup 1 \cdot P_2 \cup 1 \cdot P_3 \text{ sau}$$

$$y_1 = 0 \cdot \bar{x}_1\bar{x}_0 \cup 1 \cdot \bar{x}_1 x_0 \cup 1 \cdot x_1\bar{x}_0 \cup 1 \cdot x_1 x_0$$

Forma de exprimare a unei funcţii logice prin suma logică a mintermilor se mai numeşte forma disjunctiv-normal-perfectă (FDNP).*

În mod similar, o funcţie logică se mai poate exprima şi sub forma conjunctiv-normal-perfecta (FCNP)**, efectuînd conjuncţia tuturor sumelor logice S_i (în care intervin toate variabilele, eventual unele negate) pentru care funcţia logică ia valoarea 0.

Suma logică $S_i = x_0 \cup x_1 \cup \ldots \cup x_i \cup \bar{x}_{i+1} \cup \ldots \cup x_{n-1}$ mai poartă numele de maxterm sau constituent al zeroului. El ia valoarea zero numai pentru disjuncţia de rang i a tuturor variabilelor, în care variabila x va fi fără negaţie, dacă valoarea ei binară, în setul dat, este 0 şi va fi negată în caz contrar.

În cazul general FCNP a unei funcţii logice se mai poate scrie:

$$y = \bigcup_{i=0}^{2^n-1} (a_i \cup S_i)$$

în care a_i ia valoarea 0 sau 1, după cum funcţia y are valoarea 0 sau 1, pentru setul i de variabile.

În cazul conjuncţiei a două variabile rezultă:

$$y_2 = (0 \cup x_1 \cup x_0)(0 \cup x_1 \cup \bar{x}_0)(0 \cup \bar{x}_1 \cup x_0)(1 \cup \bar{x}_1 \cup \bar{x}_0)$$

sau:

$$y_2 = (x_1 \cup x_0)(x_1 \cup \bar{x}_0)(\bar{x}_1 \cup x_0),$$

deoarece: $1 \cup \bar{x}_1 \cup \bar{x}_0 = 1$.

Întrucît în FDNP şi FCNP fiecare din cei 2^n coeficienţi a_i (n este numărul de variabile independente) poate lua valorile 0 sau 1, vor exista în total 2^{2^n} funcţii de n variabile. Aceste funcţii se notează cu f_k, unde k este valoarea zecimală a numărului format din coeficienţii:

$$a_{2^n-1}\ a_{2^n-2} \ldots\ldots a_1 a_0$$

Fie funcţia f dată prin tabela de mai jos:

x_2	x_1	x_0	f
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

*, ** Aceste forme se numesc perfecte deoarece conţin produse sau sume logice în care intră toate variabilele. Ele se numesc normale întrucît semnul negaţiei nu se extinde decît la expresii formate dintr-o singură variabilă.

În FDNP ea se va scrie astfel:

$$f=\bar{x}_2\bar{x}_1x_0 \cup \bar{x}_2x_1\bar{x}_0 \cup x_2\bar{x}_1x_0 \cup x_2x_1x_0$$

în timp ce în FCNP, va avea următorul aspect:

$$f=(x_2 \cup x_1 \cup x_0)(x_2 \cup \bar{x}_1 \cup \bar{x}_0)(\bar{x}_2 \cup x_1 \cup x_0)(\bar{x}_2 \cup \bar{x}_1 \cup x_0).$$

În tabelele 4.2 și 4.3 sînt prezentate funcțiile logice de o variabilă și de două variabile.

Tabela 4.2

FUNCȚII LOGICE DE O VARIABILĂ

f_k \ x	0	1	Reprezentare	Denumire
f_0	0	0	0	Constanta 0
f_1	0	1	x	Variabila x
f_2	1	0	$\bar{x}$	Negația lui x
f_3	1	1	1	Constanta 1

Tabela 4.3

FUNCȚII LOGICE DE DOUĂ VARIABILE

x_1	0	0	1	1		
x_0	0	1	0	1	Reprezentare	Denumire
f_k \ a_i	a_0	a_1	a_2	a_3		
f_0	0	0	0	0	0	Constanta 0
f_1	0	0	0	1	$x_1 \cdot x_0$	Conjuncția „SI"
f_2	0	0	1	0	$x_1 \bigtriangledown x_0$	Interdicția după x_0
f_3	0	0	1	1	x_1	Variabila x_1
f_4	0	1	0	0	$x_0 \bigtriangledown x_1$	Interdicția după x_1
f_5	0	1	0	1	x_0	Variabila x_0
f_6	0	1	1	0	$x_1 \oplus x_0$	Suma modulo 2 (SAU-EX)
f_7	0	1	1	1	$x_1 \cup x_0$	Disjuncția „SAU"
f_8	1	0	0	0	$x_1 \uparrow x_0$	SAU — NU (NOR)
f_9	1	0	0	1	$x_1 \equiv x_0$	Identitatea logică
f_{10}	1	0	1	0	$\bar{x}_0$	Negația lui x_0
f_{11}	1	0	1	1	$x_0 \rightarrow x_1$	Implicația
f_{12}	1	1	0	0	$\bar{x}_1$	Negația lui x_1
f_{13}	1	1	0	1	$x_1 \rightarrow x_0$	Implicația
f_{14}	1	1	1	0	$x_1 \mid x_0$	ȘI — NU (NAND)
f_{15}	1	1	1	1	1	Constanta 1

4.3. Identități și simplificări

Formele canonice, în mod frecvent, nu reprezintă expresiile cele mai simple ale unei funcții logice. Spre exemplu, forma canonică a disjuncției (FDNP) a două variabile este:

$$\bar{x}_1 x_0 \cup x_1 \bar{x}_0 \cup x_1 x_0 \text{ și nu } x_1 \cup x_0, \text{ conform definiției.}$$

Expresiile logice se pot simplifica folosind identitățile date mai jos:

1. $x \cup 0 = x$	10. $x_1 \cup x_0 = x_0 \cup x_1$
2. $x\,0 = 0$	11. $x_1 \cdot x_0 = x_0 \cdot x_1$
3. $x \cup 1 = 1$	12. $(x_2 \cup x_1) \cup x_0 = x_2 \cup (x_1 \cup x_0)$
4. $x \cdot 1 = x$	13. $(x_2 \cdot x_1) x_0 = x_2 (x_1 \cdot x_0)$
5. $x \cup x = x$	14. $x_0 (x_1 \cup x_2) = x_0 \cdot x_1 \cup x_0 \cdot x_2$
6. $x \cdot x = x$	15. $x_0 \cup (x_1 \cdot x_2) = (x_0 \cup x_1)(x_0 \cup x_2)$
7. $x \cup \bar{x} = 1$	16. $\overline{x_1 x_0} = \bar{x}_1 \cup \bar{x}_0$
8. $\overline{(\bar{x})} = x$	17. $\overline{x_1 \cup x_0} = \bar{x}_1 \bar{x}_0$

Identitățile 1—9 sînt evidente. Identitățile 10—15 exprimă proprietățile de comutativitate, asociativitate și distributivitate ale funcțiilor logice. Identitățile 16 și 17 poartă numele de legile lui De Morgan. Ele se pot demonstra simplu cu ajutorul tabelelor de adevăr, arătînd că, pentru aceleași valori ale variabilelor x_0 și x_1, termenii stîng și drept, din egalitățile 16 și 17, iau valori egale.

4.4. Realizarea fizică a funcțiilor logice

4.4.1. Circuite logice combinaționale.

În capitolul referitor la elementele de aritmetică pentru calculatoarele numerice s-a văzut că operațiile aritmetice de adunare și înmulțire în binar se realizează pe baza unor reguli foarte simple, concretizate în tabele corespunzătoare.

Operațiile de adunare și înmulțire a două numere binare x_i și y_i, de cîte un rang, pot fi descrise cu ajutorul următoarelor reguli, unde: S_i este rangul i al sumei, T_{i+1} este transportul în rangul $i+1$ al sumei, P_i este rangul i al produsului:

a) adunarea:

— dacă x_i şi y_i sînt zero, suma S_i este zero, transportul T_{i+1} este zero;

— dacă x_i sau y_i este unu (nu simultan), suma S_i este unu, transportul Ti_{+1} este zero;

— dacă x_i şi y_i sînt unu, suma S_i este zero, transportul T_{i+1} este unu.

b) înmulţirea:

— dacă x_i sau y_i este zero, produsul P_i este zero;

— dacă x_i şi y_i sînt unu produsul P_i este unu.

Sub formă de tabelă, regulile de mai sus devin:

x_i	y_i	S_i	T_{i+1}
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

x_i	y_i	P_i
0	0	0
0	1	0
1	0	0
1	1	1

Folosind cunoştinţele din paragraful precedent rezultă că:

$$S_i = x_i \oplus y_i = \bar{x}_i y_i \cup x_i \bar{y}_i$$
$$T_{i+1} = x_i y_i$$
$$P_i = x_i y_i$$

Se poate constata uşor că operaţiile de adunare şi înmulţire binare se reduc la operaţii logice asupra argumentelor date.

Presupunînd că potenţialul electric de +5 V reprezintă unu, iar cel de 0V reprezintă zero, se poate construi un circuit electronic, cu două borne de intrare (x_i, y_i) şi două borne de ieşire (S_i, T_{i+1}), capabil să implementeze operaţia de adunare a două cifre binare, materializate prin niveluri de potenţial. Un asemenea circuit sumator se poate sintetiza cu ajutorul circuitelor electronice, care implementează diferite funcţii logice.

În practică au fost realizate, în diverse tehnologii (relee electromagnetice, elemente feromagnetice, elemente cu fluid, elemente optice, tuburi electronice, dispozitive semiconductoare discrete, circuite integrate) circuite care pot implementa funcţiile logice prezentate în tabelele 4.2 şi 4.3

Elementele constructive ale calculatoarelor electronice sînt realizate cu ajutorul circuitelor logice integrate.

Analiza şi sinteza schemelor care intră în componenţa calculatoarelor electronice, se pot efectua folosind aparatul logicii matematice. În această privinţă este concludentă stabilirea ecuaţiilor logice, care descriu funcţionarea sumatorului elementar.

Circuitele logice realizate sub formă integrată au practic una sau mai multe intrări şi pot implementa diferite funcţii logice.

Variabilele de intrare (x, y) şi ieşire (z) vor fi materializate prin niveluri de tensiune discrete: 0V şi +5V, cărora li se asociază valorile logice „0" şi respectiv „1".

Deoarece aceste circuite nu posedă memorie, ieşirile lor pot fi interpretate ca rezultate ale unor operaţii logice, atîta timp cît sînt prezente (active) semnalele de intrare. Asemenea circuite poartă numele de circuite combinaţionale.

În tabelul 4.4 se prezintă cîteva circuite logice integrate realizate în tehnologia tranzistor-tranzistor (TTL)*.

Tabela 4.4

EXEMLPE DE CIRCUITE LOGICE INTEGRATE – TTL

Circuit logic	Nr. de circuite pe pastilă	Simbol pentru elementul de bază
SI (AND) 7408	4/2	$z = x.y$
SAU (OR) 7432	4/2	$z = x \cup y$
ȘI – NU (NAND) 7400	4/2	$z = \overline{xy}$
SAU – NU (NOR) 7402	4/2	$z = \overline{x \cup y}$
NU (NOT) 7404	6/1	$z = \bar{x}$
SAU – EXCLUSIV (SUMA MODULO 2) (XOR) 7486	4/2	$z = x \oplus y$

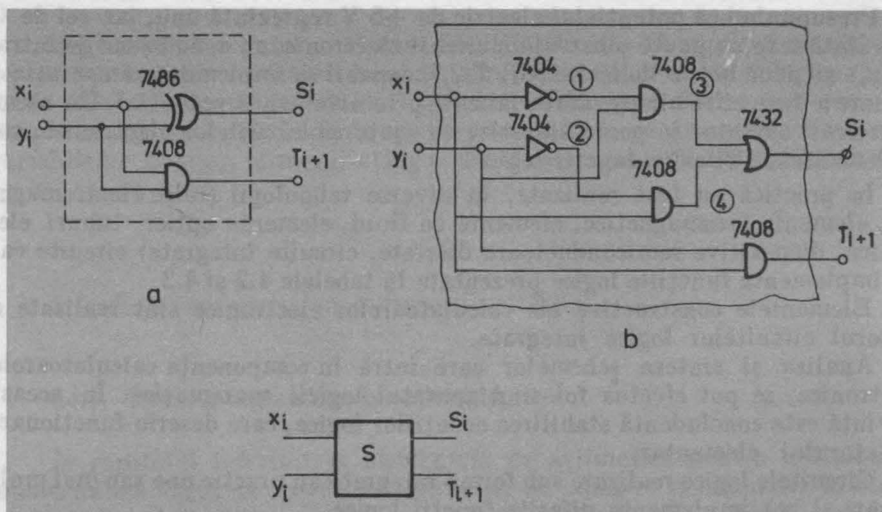


Fig. 4.1. Sumatorul cu două intrări: a) implementare cu circuitele 7486 și 7408; b) implementare cu circuitele 7404, 7408, 7432; c) schema bloc

* Există mai multe tehnologii de realizare a circuitelor logice integrate: TTL (Logica Tranzistor-Tranzistor), MOS (Metal-Oxid-Semiconductor) în variantele: PMOS (MOS canal P), NMOS (MOS canal N), CMOS (MOS complementar) etc.

Cu ajutorul unor asemenea circuite se pot implementa practic uşor diferite funcţii logice. De exemplu, plecînd de la ecuaţiile sumatorului se pot obţine implementările din figurile 4.1. a şi b, care sub forma generală se pot reprezenta ca în figura 4.1 c.

Se constată că implementarea din figura 4.1. a este mai simplă, însă circuitul din figura 4.1 b poate furniza în punctele 1—4 rezultatele $\bar{x}_i$, $\bar{y}_i$, $\bar{x}_i y_i$ şi $x_i \cdot \bar{y}_i$, care pot fi utilizate în diverse scopuri, în cadrul schemelor de prelucrare a informaţiei.

Pentru realizarea efectivă a adunării a doi biţi, trebuie să se ţină seama, pe lîngă intrările de date x_i, y_i, şi de intrarea de transport T_i, din etajul precedent, conform tabelei de adevăr pentru sumatorul complet.

T_i	x_i	y_i	S_i	T_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Prin examinarea tabelei de mai sus, se constată că un sumator complet cu trei intrări (x_i, y_i, T_i) se poate obţine cu ajutorul a două sumatoare simple S şi a unui circuit SAU (Fig. 4.2. a) Aceasta se realizează adunînd mai întîi x_i

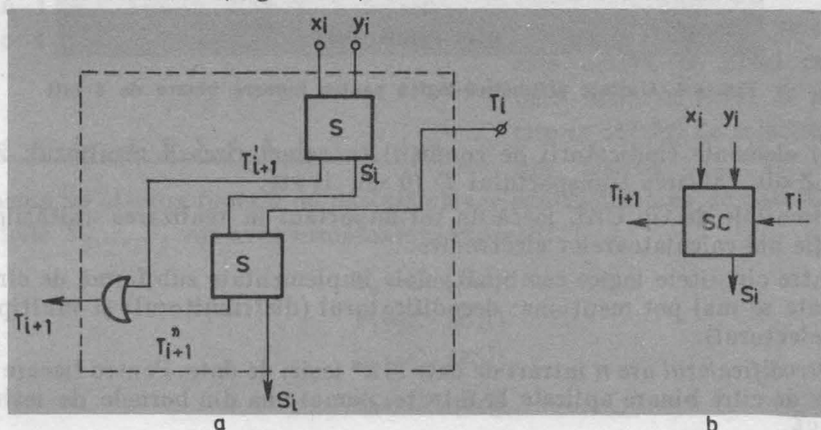
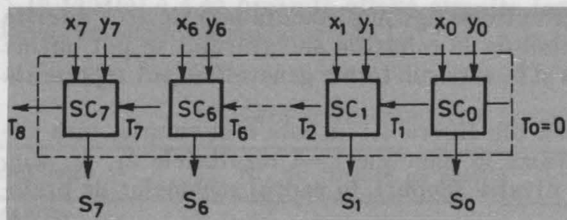


Fig. 4.2. Sumatorul cu trei intrări: a) sumator cu trei intrări realizat cu două sumatoare cu două intrări; b) schema bloc

şi y_i pentru a obţine rezultatele intermediare S_i' şi T_{i+1}', după care S_i' se va aduna cu T_i, obţinîndu-se suma corectă S_i şi transportul T_{i+1}''. Acesta din urmă va fi adunat logic cu T_{i+1}', pentru a se obţine transportul real T_{i+1}. Prin legarea în cascadă a mai multor sumatoare complete, se obţin sumatoare pentru numere binare de mai mulţi biţi. În figura 4.3 se prezintă un sumator paralel, pentru



numere binare de 8 biţi. Prin completarea schemei sumatorului cu circuite logice suplimentare, asupra operanzilor binari $X_{7:0}$ şi $Y_{7:0}$ se pot efectua şi alte operaţii aritmetice sau logice.

Fig. 4.3. Sumatorul paralel pentru numere binare de 8 biţi

Selecţia operaţiilor se va face cu un număr binar, cu mai multe ranguri, care se va aplica la un grup de intrări de comandă, înainte de forţarea operanzilor în sumator.

În acest mod se obţine o Unitate Aritmetică Logică (UAL).

În figura 4.4 se prezintă o UAL pentru numere binare de 4 biţi, asupra cărora se pot efectua diferite operaţii aritmetice/logice ale căror coduri de operaţie se aplică la intrările de comandă. La ieşire se obţin, pe lîngă rezultatul

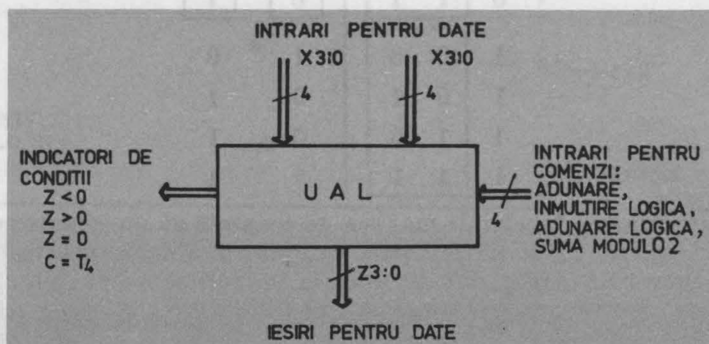


Fig. 4.4. Unitate aritmetică-logică pentru numere binare de 4 biţi

$Z_{3:0}$ şi elemente (indicatorii de condiţii) ce caracterizează rezultatul: $Z<0$, $Z>0$, $Z=0$, valoarea transportului T (0 sau 1) etc.

Circuitele de tip UAL joacă un rol important în realizarea unităţilor de execuţie ale calculatoarelor electronice.

Între circuitele logice combinaţionale implementate sub formă de circuite integrate se mai pot menţiona: decodificatorul (distribuitorul) şi multiplexorul (selectorul).

Decodificatorul are n intrări de date şi 2^n ieşiri de date. Pentru fiecare combinaţie de cifre binare aplicate la intrare, numai una din bornele de ieşire va fi activă.

Decodificatorul 7442 (fig. 4.5) este un decodificator special binar-zecimal. El dispune de 4 intrări, notate cu D, C, B, A, şi 10 ieşiri, notate cu Y_9, Y_8,..., Y_0. Pentru fiecare din numerele binare 000—1001, aplicate la intrare, numai cîte una din ieşirile Y_0,...Y_9 va fi activă, pe nivel de tensiune coborît (fapt specificat prin ieşirea marcată cu un cerc), celelalte fiind la niveluri ridicate de tensiune.

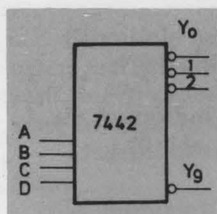


Fig. 45. Decodificator. binar-zecimal

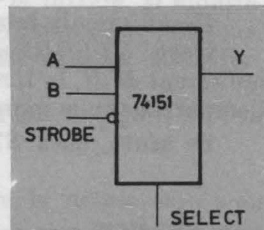


Fig. 4.6. Multiplexor 2 : 1

Tabela 4.5

TABELA DE LUCRU PENTRU DECODIFICATORUL 7442

D	C	B	A	Y									
				0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	1	1	1	1	1	1	1	1	1
0	0	0	1	1	0	1	1	1	1	1	1	1	1
0	0	1	0	1	1	0	1	1	1	1	1	1	1
0	0	1	1	1	1	1	0	1	1	1	1	1	1
0	1	0	0	1	1	1	1	0	1	1	1	1	1
0	1	0	1	1	1	1	1	1	0	1	1	1	1
0	1	1	0	1	1	1	1	1	1	0	1	1	1
0	1	1	1	1	1	1	1	1	1	1	0	1	1
1	0	0	0	1	1	1	1	1	1	1	1	0	1
1	0	0	1	1	1	1	1	1	1	1	1	1	0
.	.	.	.	.	.	.	.	.	.	.	.	.	.
1	0	1	0	1	1	1	1	1	1	1	1	1	1
1	0	1	1	1	1	1	1	1	1	1	1	1	1
.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.
1	1	1	1	1	1	1	1	1	1	1	1	1	1

Dacă intrarea *D* este tratată ca intrare de comandă, activă pe nivel coborît, decodificatorul 7442 poate fi considerat ca avînd trei intrări de date (*C*, *B*, *A*) şi opt ieşiri de date (Y_0,.. Y_1, Y_7) operînd numai în zona marcată cu linie întreruptă în tabela de adevăr 4.5. Cu condiţia $D=0$, ieşirile Y_0, Y_1, ..., Y_7 vor fi date de relaţiile:

$$Y_0 = C \cup B \cup A$$

$$Y_1 = C \cup B \cup \bar{A}$$

....

$$Y_7 = \bar{C} \cup \bar{B} \cup \bar{A}$$

Dacă intrarea *D*, de comandă, este activă pe nivel coborît şi dacă intrările *A* şi *B* sînt folosite ca intrări de selecţie a uneia din ieşirile Y_0, Y_1, Y_2, Y_3, schema va efectua funcţia de demultiplexor pentru intrarea de date *C*. Astfel, ieşirile Y_0,.., Y_3 vor avea următoarele expresii:

$$Y_0 = C \cdot (\bar{B}\bar{A})$$

$$Y_1 = C \cdot (\bar{B}A)$$

$$Y_2 = C \cdot (B\bar{A})$$

$$Y_3 = C \cdot (BA)$$

Multiplexorul este un circuit cu mai multe intrări de date şi cu o singură ieşire de date. Cu ajutorul altor intrări, de selecţie, se poate alege intrarea a cărei valoare se va atribui ieşirii.

În figura 4.6 se prezintă multiplexorul 74157, cu două intrări de date *A*, *B*, o ieşire de date *Y*, o intrare de comandă *STROBE* (activă pe nivelul coborît) şi o intrare de selecţie SELECT.

Pe un circuit integrat sînt plasate patru asemenea multiplexoare.

Dacă $STROBE=0$, (activ), stabilirea lui $SELECT=0$, va face $Y=A$, indiferent de B, în timp ce $SELECT=0$, va conduce la $Y=B$, indiferent de A. Compartimentele marcate cu* specifică condiţia indiferentă (1 sau 0).

Pe scurt, dacă $STROBE=0$, rezultă:

$$Y=A\cdot\overline{SELECT}\cup B\cdot SELECT.$$

unde SELECT este considerată variabilă logică (Tab. 4.6).

Tabela 4.6

TABELA DE LUCRU A MULTIPLEXORULUI 74157

intrări				ieşire
STROBE	SELECT	A,	B	Y
1	*	*	*	0
0	0	0	*	0
0	0	1	*	1
0	1	*	0	0
0	1	*	1	1

4.4.2. Circuite logice secvenţiale.

În funcţionarea reală a circuitelor combinaţionale timpul poate interveni în sensul întîrzierii apariţiei semnalelor de ieşire faţă de momentul aplicării semnalelor de intrare. Aceasta se datorează valorii finite a timpului de propagare a semnalelor, prin circuitele logice din care este construită schema dată.

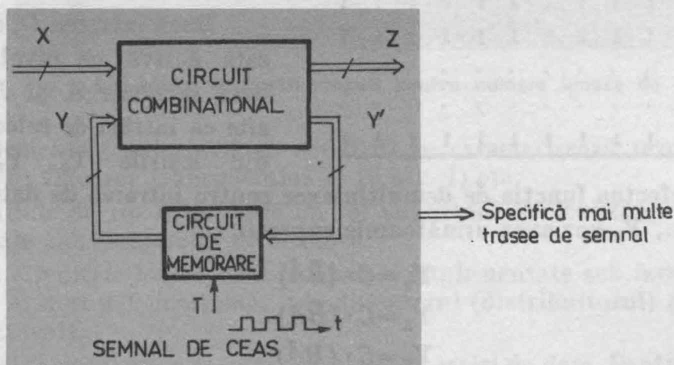


Fig. 4.7. Modelul general al unui circuit secvenţial.

Completînd, în mod corespunzător, o schemă constituită din circuite combinaţionale, cu elemente capabile să memoreze informaţia, se pot obţine circuite noi, numite secvenţiale.

În circuitele secvenţiale, ieşirile la un moment dat nu depind numai de intrările aplicate la acel moment, ci şi de intrările aplicate la momentele anterioare.

Circuitele secvenţiale la care aplicarea semnalelor la intrare şi modificarea semnalelor de ieşire se realizează sub comanda unui semnal de sincronizare (tact), poartă numele de circuite secvenţiale sincrone. Acestea sînt frecvent folosite în calculatoare.

Modelul general al unui circuit secvenţial sincron este dat în figura 4.7, în care:

X reprezintă intrările curente de date.

Y constituie intrări ce conţin informaţii referitoare la datele care s-au aplicat la momentele anterioare,

Z este ieşirea curentă,

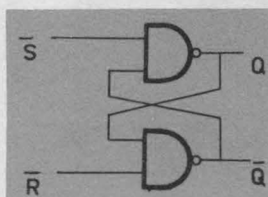
Y' reprezintă informaţia care va fi furnizată la intrarea Y a circuitului, în ciclul următor de lucru; ea este generată pe baza „istoriei" circuitului şi a informaţiei curente de la intrarea X.

Se constată că circuitul trebuie să funcţioneze secvenţial, sub controlul unui semnal de sincronizare T, asociat cu timpul.

Informaţia referitoare la timp (T) *este furnizată sub forma discretă — impulsuri*, care apar la momente bine definite în timp, purtînd numele de semnale de ceas sau simplu „ceas".

Un asemenea circuit secvenţial este un circuit sincron.

Elementele de memorie poartă numele de circuite bistabile. Ele pot fi realizate fizic sub forme diferite, utilizînd elemente logice SI-NU, SAU-NU, în conjuncţie cu inversoare etc. În figura 4.8 se prezintă un bistabil realizat cu circuite ŞI-NU (NAND) şi tabela corespunzătoare de adevăr.



	Q^n	$\bar{S}$	$\bar{R}$	Q^{n+1}	$\bar{Q}^{n+1}$
*	0	0	0	1	1
	0	0	1	1	0
	0	1	0	0	1
	0	1	1	0	1
*	1	0	0	1	1
	1	0	1	1	0
	1	1	0	0	1
	1	1	1	1	0

* Situaţie nepermisă

Fig. 4.8. Bistabil realizat cu circuite SI—NU (NAND) şi tabela de adevăr

El posedă intrările $\bar{S}$, $\bar{R}$ şi ieşirile Q şi $\overline{Q}$. Q^n reprezintă ieşirea curentă la momentul (pasul) n, al secvenţei, iar Q^{n+1} este ieşirea la pasul următor, al secvenţei.

Funcţionarea lui se poate urmări considerînd că, iniţial, sînt îndeplinite condiţiile:

$$Q=0;\ \bar{S}=1;\ \bar{R}=1,$$

care reprezintă o stare stabilă, caracterizată prin ieşirea $Q=0$. Dacă $\bar{S}=0$ şi $\bar{R}=1$, rezultă $Q=1$. În continuare, dacă $\bar{S}=1$ şi $\bar{R}=1$, Q se va menţine egal cu 1.

Dacă $\bar{S}=1$ şi $\bar{R}=0$, rezulta $Q=0$. În continuare, dacă $\bar{S}=1$ şi $\bar{R}=1$, Q se va menţine egal cu 0.

Nu este permis ca ambele intrări $\bar{S}$ şi $\bar{R}$ să fie zero simultan, deoarece ieşirile Q şi $\bar{Q}$ vor fi forţate la nivel ridicat (unu), în timp ce ele, conform definiţiei, trebuie să aibe valori complementare.

Se constată uşor că ecuaţia de funcţionare a circuitului va fi:

$$Q^{n+1}=\bar{S}\cup\bar{R}\cdot Q^{n}$$

Ieşirea Q^{n+1} la pasul următor al secvenţei este funcţie de valorile curente ale semnalelor S, $\bar{R}$ şi de valoarea Q^n, memorată la pasul curent.

Pentru a asigura controlul prin semnalul de ceas, în vederea stabilirii cu precizie a momentului memorării informaţiei, se va folosi elementul bistabil din figura 4.9. Acesta posedă o intrare de date, la care se aplică informaţia binară ce trebuie memorată la forţarea semnalului de ceas. Memorarea se efectuează cînd semnalul de ceas se afla pe nivel ridicat (valoarea logică unu).

Ecuaţia de funcţionare a circuitului bistabil (de tip D) este următoarea:

$$Q^{n+1}=C\cdot D\cup\bar{C}\cdot Q^{n}$$

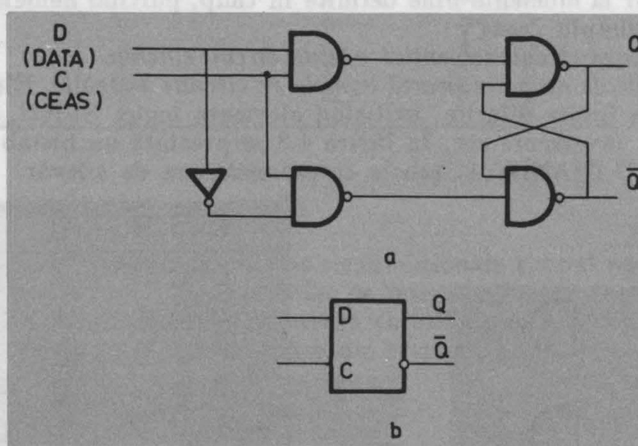


Fig. 4.9. Bistabil de tip D: a) schema logică; b) simbol

Cînd semnalul de ceas este la nivel coborît (0), $Q^{n+1}=Q^{n}$, indiferent de intrarea D; dacă semnalul de ceas este la nivel ridicat (1), $Q=D$, indiferent de vechea valoare Q.

Registre şi numărătoare-aplicaţii.

Registre. Folosind asamblaje de elemente bistabile (de exemplu de tip D) se pot implementa scheme de memorare a informaţiei binare, sub formă de cuvinte cu mai multe ranguri.

În figura 4.10*a* se prezintă un registru notat cu A, cu patru ranguri, cu intrările de date $X_{3:0}$, intrarea de ceas C şi ieşirile $A_{3:0}$, iar în figura 4.10*b* se prezintă notaţia simplificată.

Pentru a specifica procesul de încărcare a cuvîntului binar $X_{3:0}$ în registrul $A_{3:0}$, la momentul aplicării semnalului de ceas C, se poate folosi notaţia:

$$C\cdot A\leftarrow X;$$

---

* Există şi alte tipuri de bistabile: J K, T, RS etc.

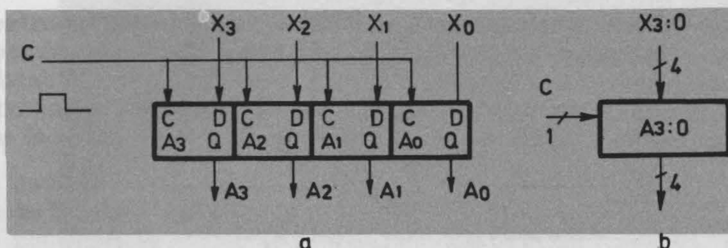


Fig. 4.10. Registru cu patru ranguri, cu încărcare paralelă: a) schema extinsă; b) simbol

Registrul A poate avea mai multe surse de informaţii. De exemplu, acestea pot fi alte registre sau ieşiri ale unor scheme combinaţionale, notate cu $X_{3:0}$, $Y_{3:0}$, $Z_{3:0}$. Dacă se urmăreşte încărcarea succesivă a cuvintelor X, Y, Z în registrul A, la semnalele de ceas C_i, C_{i+1}, C_{i+2}, se poate imagina schema de principiu, din figura 4.11.

Operaţiile se pot descrie sub forma următoare:

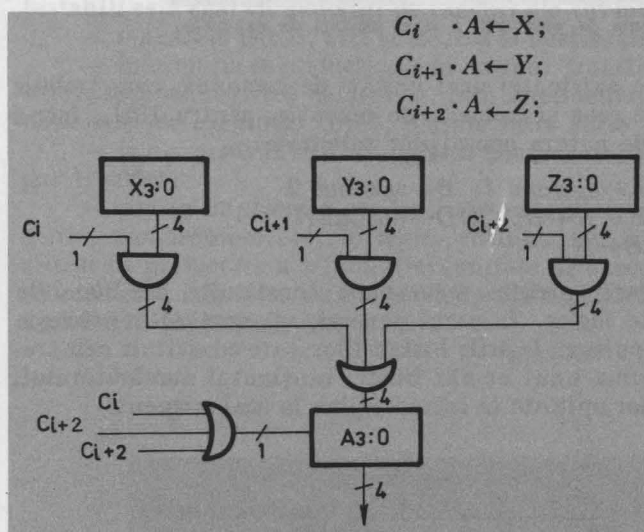


Fig. 4.11. Încărcarea unui registru de la mai multe surse de informaţii

O Unitate Aritmetică Logică (*UAL*) poate fi completată cu două registre de intrare (A, B), pentru operanzi şi cu un registru de ieşire (R), pentru rezultat.

Se considera că:

— la semnalul de cesa C_1 se încarcă operanzii în registrele A şi B

— la semnalul C_2 are loc operaţia de adunare prin activarea unui cod corespunzător de operaţie la *UAL*, încărcarea registrului R, cu rezultatul obţinut şi a registrului *CND*, cu indicatorii de condiţii.

În figura 4.12 este prezentată schemă de principiu a unei unităţi simple de execuţie. Astfel, s-a implementat o schemă simplă de coduri de operaţii, cu ajutorul cărora se pot efectua operaţiile de: adunare, înmulţire logică şi sumă logică asupra a doi operanzi, reprezentaţi prin cuvinte binare de 4 biţi.

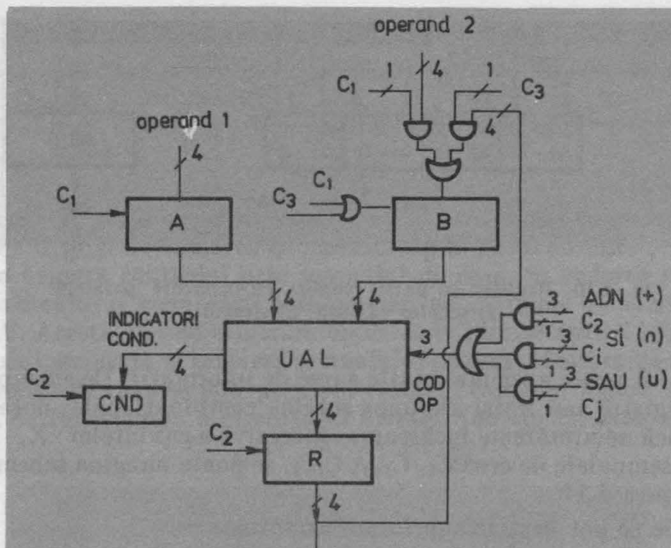


Fig. 4.12. Schema de principiu a unei unităţi de execuţie

Se constată necesitatea existenţei unei unităţi de comandă, care trebuie să distribuie semnalele de ceas şi codurile de operaţie, pentru UAL, într-o secvenţă dată, în funcţie de natura operaţiilor solicitate:

C_1. *A←operand 1*; *B←operand* 2
C_2. *R←A+B*; *CND←indicatori*
C_3. *B←R*

Numărătoarele reprezintă circuite secvenţiale constituite din bistabile interconectate prin circuite logice. În cazul general, un numărător primeşte la intrarea sa o serie de impulsuri. Ieşirile bistabililor, care constituie numărătorul, vor reflecta, sub forma unui număr binar, conţinutul numărătorului, egal cu numărul impulsurilor aplicate la intrare, pînă în acel moment.

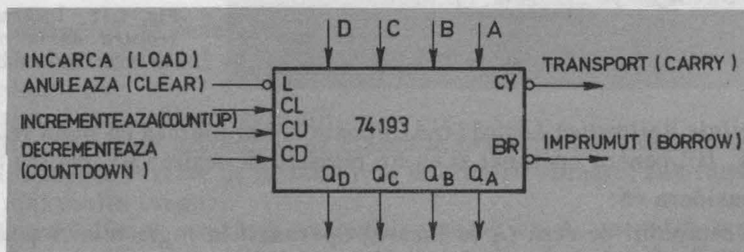


Fig. 4.13. Terminalele numărătorului 74193

Un numărător cu 4 ranguri numără maximum 16 impulsuri, înainte de a reveni în starea iniţială. Dacă, iniţial, conţinutul său era 0000, el va număra în secvenţa: 0001, 0010, 0011,...., 1110, 1111, 0000. Se spune că acest numărător operează „modulo 16".

Pentru a utiliza un numărător este necesar ca el să mai posede şi o intrare de comandă care, la primirea unui semnal, va aduce în starea iniţială (zero) toţi bistabilii.

Numărătoarele realizate sub formă de circuite integrate pot dispune de o serie de facilităţi, activate prin operaţii de comandă corespunzătoare:

— anulare conţinut (*CLEAR*),
— încărcare (*LOAD*),
— numărare în sens crescător-incrementare (*COUNT-UP*),
— numărare în sens descrescător-decrementare (*COUNT-DOWN*)

Astfel, numărătorul 74193, pe 4 biţi, (fig. 4.13) are un caracter reversibil, necesitînd, pe lîngă semnalele de comandă amintite mai sus şi intrările paralele *A*, *B*, *C*, *D*; ieşirile paralele Q_A, Q_B, Q_C, Q_D; ieşirea de împrumut (*BORROW*) şi ieşirea de transport (*CARRY*).

Numărătoarele, împreună cu registrele, precum şi cu alte structuri logice secvenţiale constituie resursele fizice (hardware) ale calculatoarelor numerice.

Din cele prezentate mai sus, se pot observa următoarele:

— cuvintele binare (operanzii) sînt stocaţi în registre, al căror număr de bistabili este egal cu numărul de ranguri ale cuvîntului binar;
— registrele pot fi, atît surse, cît şi destinaţii de operanzi;
— informaţia se prelucrează în timpul transferului său, de la registrele sursă, spre un registru destinaţie, prin intermediul unor scheme logice combinaţionale (de exemplu: UAL), plasate între surse şi destinaţii;
— la un moment dat un registru poate fi destinaţie numai pentru un singur transfer;
— pentru implementarea unei secvenţe de transferuri între resursele (registre, numărătoare, reţele logice combinaţionale, trasee de legătură) unui sistem de prelucrare a informaţiei (unitate de execuţie), sînt necesare semnale de comandă, furnizate de o schemă specială numită unitate de comandă, care înglobează şi „ceasul" sistemului.

Partea a III-a

CALCULATORUL PERSONAL HC-85, STRUCTURĂ, COMPONENTE, OPERARE, PROGRAMARE

Capitolul 5 | Calculatoare numerice. Structură şi mod de operare.

5.1. Conceptul de calculator

Un calculator numeric este construit dintr-un ansamblu de resurse fizice (hardware) şi de programe de sistem (software de bază), care asigură prelucrarea informaţiilor în conformitate cu algoritmii specificaţi de utilizator, prin programele de aplicaţii (software de aplicaţii).

În contextul dat, un algoritm reprezintă o colecţie de reguli, o secvenţă de acţiuni elementare, privind efectuarea unor operaţii cu caracter aritmetic-logic, asupra unor date, pentru a produce alte date, reprezentînd rezultatele — datele de ieşire. Un algoritm trebuie să aibă următoarele caracteristici:

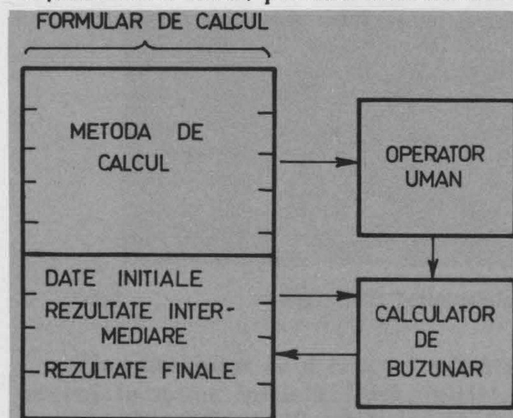


Fig. 5.1. Schema execuţiei unui algoritm de către un operator uman

— să fie finit, să se termine după un număr finit de paşi;

— să fie determinist, fără ambiguităţi — aplicarea lui repetată asupra aceloraşi date de intrare, să conducă la acelaşi rezultat;

— să aibă o intrare — un set iniţial de date;

— să aibă o ieşire — rezultatele;

— să fie eficient, în sensul că operaţiile implicate să se execute exact şi într-un interval finit de timp.

Un algoritm de calcul poate fi executat manual, de către un operator uman, care dispune de un formular de calcul ce conţine datele şi metoda de calcul (sub forma unor instrucţiuni), un creion etc. (fig. 5.1).

Operatorul va interpreta instrucţiunile care descriu metoda de calcul(programul) în termenii:

— operaţiilor aritmetice care pot fi efectuate cu calculatorul de buzunar;

— operaţiilor de decizie, privind execuţia instrucţiunii următoare din program, în funcţie de rezultatele curente;

— operaţiilor de citire a unor date şi scriere a unor rezultate în compartimentele corespunzătoare ale formularului de calcul.

Se poate observa că, sub forma cea mai generală, formularul de calcul joacă rolul unei memorii, care dispune de:

— o zonă formată din compartimente ce conţin instrucţiunile, care descriu metoda de calcul; compartimentele vor fi, în general parcurse secvenţial;

— o zonă ce conţine compartimente în care sînt plasate sau se introduc datele de intrare, rezultatele intermediare şi rezultatele finale.

Compartimentele în care sînt plasate instrucţiunile vor fi numerotate (numerele respective vor fi tratate drept adrese de instrucţiuni). Instrucţiunile vor fi executate în ordinea crescătoare a adreselor, atîta timp cît instrucţiunea curentă nu impune altă ordine de parcurgere a secvenţei de instrucţiuni — a programului.

Compartimentele în care se introduc sau în care se află deja datele de intrare şi compartimentele în care se vor plasa rezultatele finale vor fi numerotate (în continuarea compartimentelor programului), vor avea adrese.

Metoda de calcul va fi descrisă prin instrucţiuni de forma prezentată în exemplul următor:

i.	citeşte data de la adresa j
$i+1$	adună data de la adresa $j+1$
$i+2$	memorează rezultatul la adresa $j+2$
$i+3$	dacă rezultatul este mai mare decît zero, execută instrucţiunea de la adresa $i+7$
$i+4$	stop
	....
$i+7$	execută instrucţiunea i
	....
j	*data* 1
$j+1$	*data* 2
$j+2$	*data* 1+*data* 2 (după execuţia programului)

Analizînd acest program se pot constata următoarele:

— instrucţiunile şi datele sînt plasate în compartimentele aceluiaşi mediu ce joacă rol de memorie;

— instrucţiunile şi datele sînt apelate prin adrese;

— o instrucţiune conţine un cîmp care specifică operaţia de efectuat (codul de operaţie) şi un cîmp ce specifică adresa operandului (adresa);

— codurile de operaţie se referă la: citirea/stocarea datelor în memorie, efectuarea unor operaţii aritmetice (adunare, scădere, înmulţire, împărţire etc.), la modificarea secvenţei de parcurgere a instrucţiunilor programului, în funcţie de anumite condiţii sau necondiţionat (instrucţiunile de la adresele $i+3$ şi $i+7$).

5.2. Structura şi operarea calculatoarelor numerice

Un algoritm poate fi executat în mod automat cu ajutorul unui calculator. În acest scop (conform principiilor stabilite de J. von Neumann) un calculator trebuie să posede următoarele elemente:

— un mediu de intrare (*unitate de intrare — UI*) pentru instrucţiuni şi date (operanzi);

— un mediu de ieşire (*unitate de ieşire — UE*) pentru extragerea rezultatelor şi prezentarea acestora într-o formă accesibilă utilizatorului;

— o memorie (*unitatea de memorie — UM*) în care se pot stoca programul, datele iniţiale, rezultatele parţiale şi finale;

— un asamblu de prelucrare (*unitatea aritmetică — logică sau de execuţie — UAL*), capabil să efectueze operaţii aritmetice şi logice, în conformitate cu un algoritm dat, specificat prin program;

— un element de decizie şi comandă (*unitatea de comandă — UC*), care, pe baza rezultatelor parţiale obţinute, poate selecta una din opţiunile posibile de continuare a calculelor şi asigura controlul întregului ansamblu de unităţi, în vederea funcţionării lui automate.

În figura 5.2 se prezintă schema funcţională a unui calculator, constituită din unităţile specificate anterior.

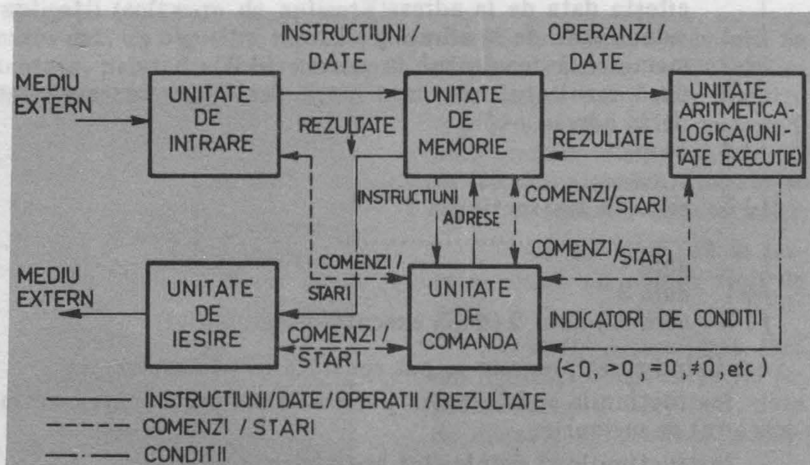


Fig. 5.2. Schema funcţională a unui calculator

Unitatea de intrare preia, sub controlul unităţii de comandă, informaţia (instrucţiuni/date) de la o serie de echipamente periferice de intrare (cititor de cartele perforate, cititor de bandă perforată, tastatura unei console operator — display), echipamente de stocare externă a informaţiei pe medii magnetice (discuri, benzi, casete, discuri flexibile) şi o aduce la o formă standard de reprezentare, din punct de vedere electric, transferînd-o, în continuare, în unitatea de memorie.

Echipamentele periferice se conectează la unitatea de intrare prin aşa-numita interfaţă standard.

Unitatea de ieşire preia, sub controlul unităţii de comandă, informaţia corespunzătoare din memorie şi o transferă unor echipamente periferice de ieşire: consolă — display, perforator de bandă, imprimată, trasator de curbe (plotter) sau unor echipamente de stocare externă a informaţiei pe medii magnetice: discuri, benzi, casete, discuri flexibile etc.

În cele mai multe cazuri unităţile de intrare/ieşire sînt prezentate împreună, sub forma unităţii (subsistemului) de intrare/ieşire (I/E) care, la unele calculatoare, mai poartă numele de *Unitate de schimburi* (Felix C-265) sau *Canal* (IBM 360/370). În figura 5.3 se prezintă, sub formă de schemă bloc unitatea de I/E.

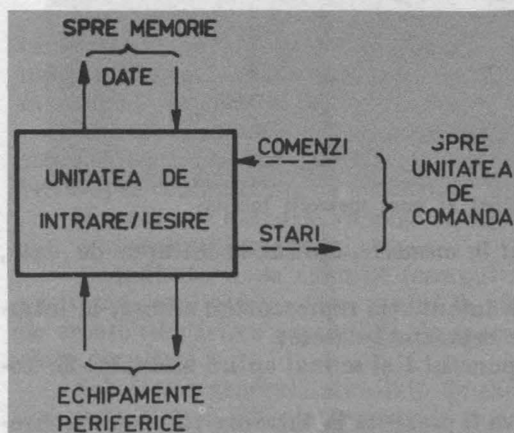


Fig. 5.3. Unitatea de intrare/ieşire

Unitatea de memorie are drept rol stocarea programelor şi a datelor. În prezent memoria se realizează cu ajutorul circuitelor electronice integrate pe scară largă, constituite din structuri elementare de memorare a informaţiei (reprezentată sub forma unor niveluri de potenţial electric), împreună cu circuitele de acces şi comandă.

Elementele de memorare (circuitele bistabile, în cazul memoriilor statice sau capacităţile de grilă ale unor inversoare, în cazul memoriilor dinamice) sînt organizate sub forma unor registre capabile să memoreze cîte un cuvînt binar. Fiecare element de memorare stochează un bit al unui cuvînt dat.

Din punct de vedere funcţional, memoria poate fi examinată pe baza modelului din figura 5.4, în care $\mathbf{M}^i$ reprezintă cuvîntul *i*, de la adresa *i*, din memorie.

Se poate observa că, pe baza adresei de *n* biţi, aplicată la intrarea decodificatorului de adrese, una din cele 2^n ieşiri (0, 1, 2, ..., 2^n-1) va fi activată, selectînd un cuvînt din memorie. Selecţia se referă atît la circuitul *ŞI1*, de aplicare a semnalului de ceas, cît şi la circuitul *ŞI2*, folosit pentru citirea conţinutului registrului respectiv

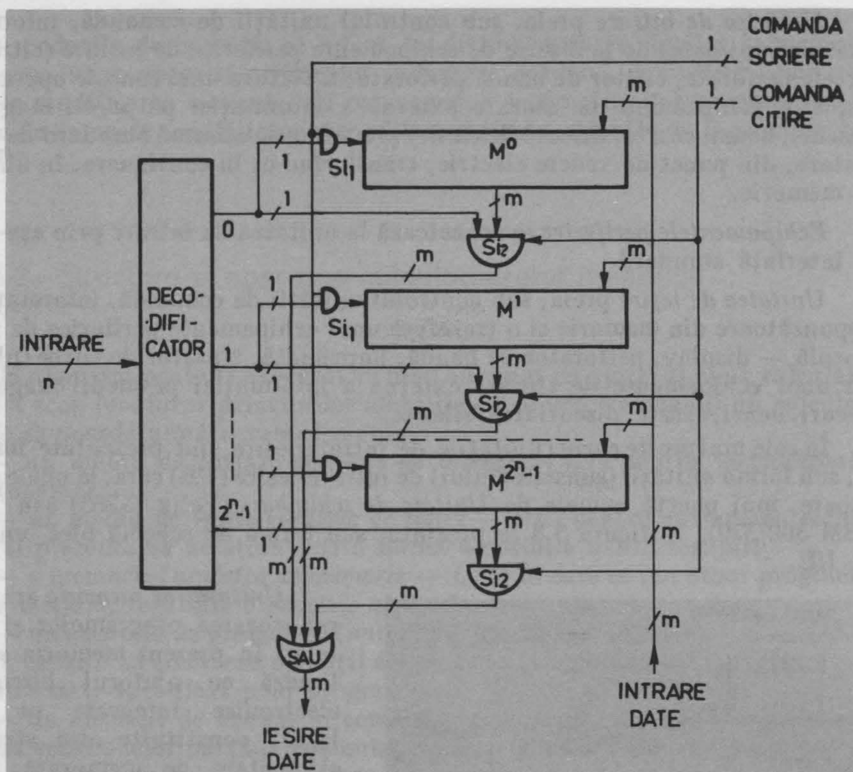


Fig. 5.4. Modelul funcţional al unei memorii interne

Operaţia de înscriere a unui cuvînt în memorie, aplicat la intrarea de date, se efectuează în următoarele etape:

1. la intrarea de adrese se aplică informaţia reprezentînd adresa, la intrarea de date se aplica informaţia care reprezintă datele;
2. se menţin informaţiile de la punctul 1 şi se mai aplică semnalul de comandă — scriere;

Este evident că informaţia, care va fi prezentă la intrarea tuturor registrelor, va fi înscrisă numai în registrul selectat de ieşirea decodificatorului şi activat de semnalul de scriere.

Operaţia de citire se va desfăşura în următoarele etape:

1. la intrarea de adrese se aplică informaţia reprezentînd adresa;
2. se menţine informaţia de la punctul 1 şi se aplică semnalul de comandă-citire.

Pe baza adresei decodificate şi a semnalului de citire, se va activa circuitul *ŞI2*, de la ieşirea registrului selectat. Acesta va permite transferarea conţinutului registrului respectiv, la intrarea circuitului *SAU*, care îl va transmite mai departe, la ieşirea sa.

În cazul examinat, numărul de biţi ai adresei este *n*, iar cel al datelor este *m*. Memoria va avea o capacitate de 2^n cuvinte binare, de cîte *m* biţi fiecare.

În mod curent capacitatea unei memorii se măsoară în Kcuvinte ($1 K = 2^{10} = 1024$). Microcalculatorul **HC-85** are o memorie cu o capacitate totală de

64 Kocteţi — cuvinte de 8 biţi, în timp ce calculatorul FELIX C-256 are o memorie cu capacitatea de 256 Ko.

O altă caracteristică a memoriei o reprezintă timpul de acces, care oferă o imagine asupra vitezei de lucru (citire sau scriere). De exemplu, o memorie cu timpul de acces de 450 ns permite scrierea sau citirea a $2 \cdot 10^6$ cuvinte/s.

Pentru aprofundarea modului de funcţionare a unui calculator, se va presupune că memoria **M** va avea un registru *RA* pentru stocarea adreselor şi un registru *RD*, pentru intrări/ieşiri de date, conform modelului din figura 5.5.

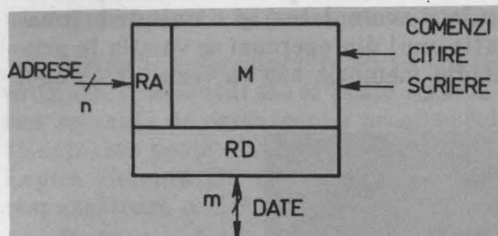


Fig. 5.5. Modelul memoriei interne cu registrele de adresă (RA) şi de date (RD)

În aceste condiţii funcţionarea memoriei poate fi descrisă astfel:

scriere:

1. $RA \leftarrow adresa$; $RD \leftarrow data$;
2. $M^{\perp RA} \leftarrow RD$;

citire:

1. $RA \leftarrow adresa$;
2. $RD \leftarrow M^{\perp RA}$

Notaţia $M^{\perp RA}$ trebuie interpretată în sensul că simbolul $\perp RA$ specifică conversia numărului binar — adresa — din *RA*, într-un număr zecimal *i*, care reprezintă numărul cuvîntului selectat, din memorie, fie în vederea citirii, fie în vederea scrierii.

Unitatea de execuţie (*Unitatea Aritmetică — Logică*) se caracterizează prin aceea că poate efectua operaţii aritmetice/logice asupra operanzilor aplicaţi la intrare, în conformitate cu o comandă, un cod de operaţie, furnizat din exterior.

Unitatea de execuţie va avea ca ieşiri:

— rezultatul operaţiei,

— indicatorii de condiţii (semnul rezultatului, rezultatul egal cu zero, paritatea rezultatului, transportul în afara rangului de semn etc), indicatorii de eroare (depăşirea capacităţii de reprezentare a numerelor în calculator, de către rezultat).

Sub forma generală o unitate de execuţie se poate reprezenta ca în figura 5.6 unde *RI*1 şi *RI*2 sînt registrele în care se plasează operanzii de prelucrat, iar *RE* este registrul de ieşire al rezultatului din unitate.

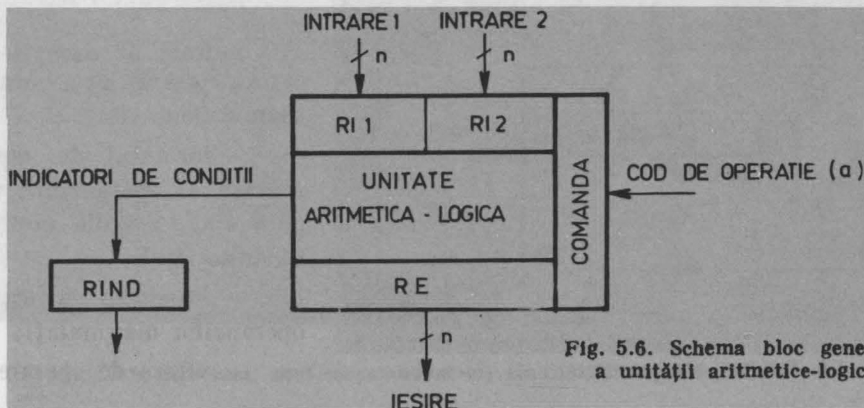


Fig. 5.6. Schema bloc generală a unităţii aritmetice-logice

Sub forma cea mai simplă operarea unității de execuție se poate descrie astfel:

$$j.\ RE \leftarrow RI1 \varrho RI2;\ RIND \leftarrow indicatori;$$

unde ϱ reprezintă una din operațiile pe care le poate efectua unitatea, iar j este semnalul de tact la care are loc operația.

O unitate de execuție modernă poate dispune de mai multe registre generale sau specializate, în care se pot afla operanzi, adrese etc.

Sub forma cea mai simplă o unitate de execuție se poate considera că este constituită dintr-un singur registru (numit și acumulator) și o unitate aritmetică — logica. (fig. 5.7). În această situație unul din operanzi se va afla în acumulator (eventual încărcat în prealabil din memorie sau ca rezultat al unei operații anterioare).

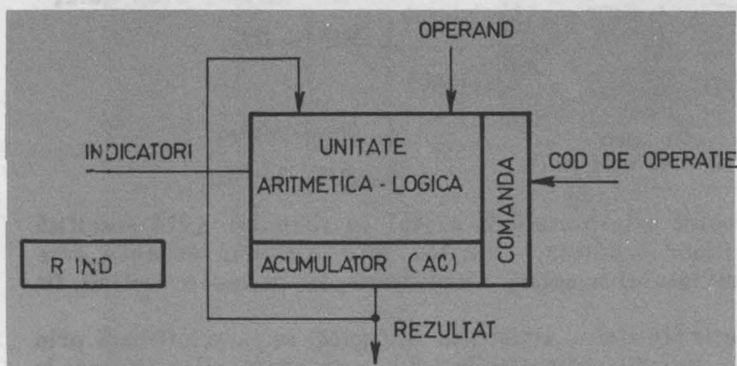


Fig. 5.7. Schema bloc a unității aritmetice-logice cu un singur acumulator

Pentru a aduna două cuvinte aflate în memorie, cu plasarea rezultatului în acumulator (AC), se va încărca, mai întîi, AC cu primul operand, după care, AC se va aduna cu cel de-al doilea operand, citit din memorie și stocat temporar în registrul de date al memoriei RD, rezultatul fiind în cele din urmă încărcat în AC:

$$j.\ AC \leftarrow AC + RD;$$

O unitate de execuție se caracterizează prin următoarele elemente:

— formatul de reprezentare al operanzilor: virgulă fixă, virgulă mobilă, binar-zecimal;

— lungimea în biți a operanzilor manipulați;

— viteza de operare.

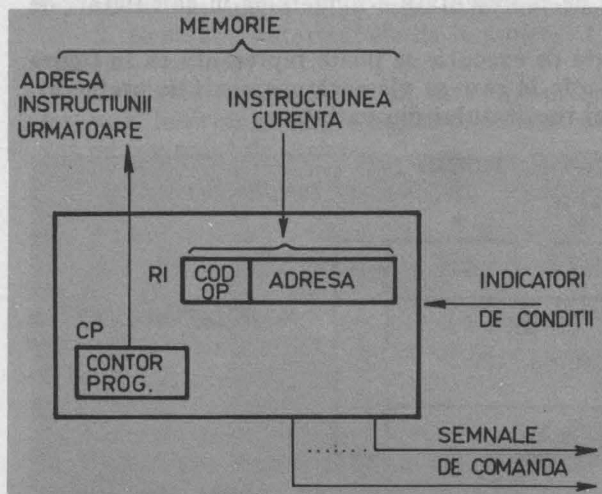


Fig. 5.8. Schema bloc a unității de comandă

Unitatea de comandă asigură citirea instrucţiunilor programului din memorie şi execuţia lor automată. Ea coordonează, prin semnale de comandă, funcţionarea tuturor celorlalte unităţi ale calculatorului.

În principiu, unitatea de comandă (*UC*) este construită dintr-un registru de instrucţiuni *RI*, un registru contor program *CP* şi logica de comandă aferentă (fig. 5.8).

Registrul instrucţiunii (*RI*) păstrează instrucţiunea curentă, citită din memorie, pe toată durata execuţiei sale. Instrucţiunea va specifica, de regulă, un cod de operaţie şi una sau mai multe adrese de operanzi.

Contorul programului (*CP*) va conţine adresa instrucţiunii următoare, din program. Conţinutul său se poate încărca direct (în cazul unor operaţii ce modifică secvenţa de parcurgere a programului: transferuri condiţionate sau necondiţionate) sau poate fi incrementat (în cazul parcurgerii secvenţiale a programului). Logica aferentă *UC* testează indicatorii de condiţii şi asigură modificarea corespunzătoare a *CP*.

Pe baza codului de operaţie *UC* furnizează semnalele de comandă pentru controlul *UI/E*, *UM*, *UE*, pe durata fiecărei instrucţiuni, în sincronism cu semnalul furnizat de un oscilator, care joacă rolul de ceas, în sistem.

Secvenţa de operaţii, referitoare la citirea unei instrucţiuni din memorie, este următoarea:

1. $RA \leftarrow CP$;
2. $RD \leftarrow M^{[RA]}$;
3. $RI \leftarrow RD$;

Contorul programului se incrementează la sfîrşitul fazei de execuţie a instrucţiunii curente, în cazul parcurgerii secvenţiale a programului.

Ansamblul unitate de comandă — unitate de execuţie poartă numele de unitate centrală de prelucrare sau procesor.

Procesorul împreună cu unitatea de memorie formează unitatea centrală a calculatorului (fig. 5.9).

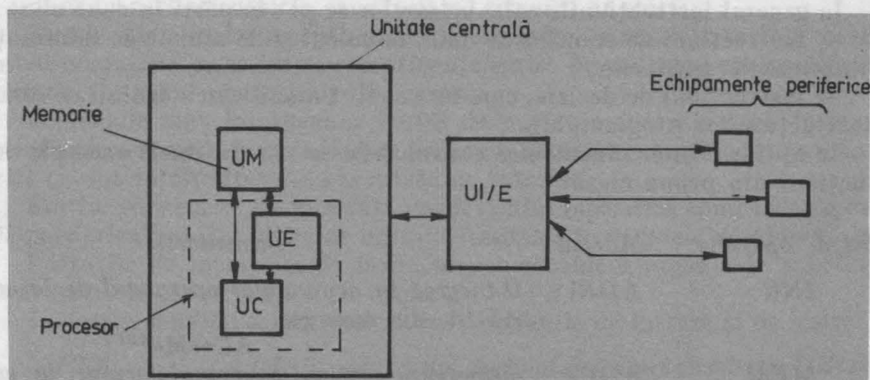


Fig. 5.9. Structura unui calculator la nivelul unităţilor funcţionale

5.3. Instrucţiunile calculatoarelor numerice

Instrucţiunile calculatoarelor numerice conţin, după cum s-a observat anterior, specificaţii referitoare la operaţia care trebuie efectuată de către o unitate dată, din componenţa calculatorului (adesea unitatea de execuţie) şi specificaţii referitoare la adresa unui operand sau a unei instrucţiuni. În unele cazuri o instrucţiune poate conţine mai multe adrese: adresa primului operand, adresa celui de-al doilea operand şi, eventual, adresa rezultatului. În cele ce urmează se vor considera instrucţiuni simple, avînd o singură adresă (fig. 5.10).

Cîmpul codului de operaţie specifică una din funcţiile ce se pot executa de către unităţile calculatorului. Dacă acest cîmp posedă m biţi, se pot codifica 2^m instrucţiuni diferite.

Cîmpul de adresă specifică o adresă de operand sau de instrucţiune. În cazul în care cîmpul de adresă conţine n biţi, se poate explora un spaţiu de adresare în memorie de 2^n cuvinte.

Totalitatea instrucţiunilor executabile de către calculatorul numeric, la nivelul la care a fost tratat, formează aşa-numitul „limbaj maşină".

Pentru a fi executate, instrucţiunile trebuie să fie transmise unităţii de comandă sub forma unor cuvinte binare (cod maşina).

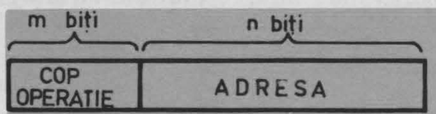


Fig. 5.10. Structura unei instrucţiuni simple la nivel de cîmpuri

În vederea simplificării muncii programatorilor, cîmpurile de cod de operaţie şi adrese se înlocuiesc prin simboluri literale mnemotehnice (mnemonice). care pot fi traduse în coduri maşină în mod automat, cu ajutorul unui program numit asamblor.

Limbajul care conţine instrucţiunile reprezentate sub formă de mnemonice poartă numele de limbaj de asamblare.

Limbajele maşina şi de asamblare sînt limbaje de nivel coborît, deoarece ele sînt interpretate direct de către calculator, la nivelul examinat, numit — nivelul maşinii convenţionale. Cu ajutorul lor se scriu programele de sistem, necesare exploatării eficiente a resurselor fizice ale calculatorului.

În general instrucţiunile calculatoarelor se pot împărţi în două clase:

— instrucţiuni de transfer de date, de calcul şi de schimb de informaţii cu echipamentele periferice,

— instrucţiuni de decizie, care au ca efect modificarea ordinii de execuţie a instrucţiunilor programului.

Cu ajutorul unor mnemonice convenabile se vor da unele exemple de instrucţiuni din prima clasă:

Cod de operaţie	*Adresă*	*Comentarii,*
INC	*ADR*1;	*O încarcă în acumulator operandul de la adresa ADR1, din memorie:* $AC \leftarrow M^{ADR1}$
MEM	*ADR2*;	*memorează conţinutul acumulatorului, la adresa sa ADR2:* $M^{ADR2} \leftarrow AC$

ADN *ADR*3; *adună conținutul acumulatorului cu conținutul celulei de memorie de la adresa ADR3 și plasează rezultatul în acumulator:*

$$AC \leftarrow AC + M^{\perp ADR3}$$

SCD *ADR*4; *scădere:*

$$AC \leftarrow AC - M^{\perp ADR4}$$

ADL *ADR*5; *adunare logică, bit cu bit:*

$$AC \leftarrow AC \cup M^{\perp ADR5}$$

INL *ADR*6; *înmulțire logică, bit cu bit:*

$$AC \leftarrow AC \cap M^{\perp ADR6}$$

SMD *ADR*7; *suma modulo doi:*

$$AC \leftarrow AC \oplus M^{\perp ADR7}$$

Instrucțiunile care se referă la conținutul acumulatorului, nu necesită specificarea adresei:

NEG ; *conținutul acumulatorului se inversează bit cu bit:*

$$AC \leftarrow \overline{AC}$$

DES; *conținutul acumulatorului se deplasează spre stînga cu un bit:*

$$AC_{0:7} \leftarrow AC_{1:7}, 0$$

DED; *conținutul acumulatorului se deplasează spre dreapta cu un bit, cu extensia rangului de semn:*

$$AC_{0.7} \leftarrow AC_0,\ AC_{0:6}$$

RST; *conținutul acumulatorului se rotește spre stînga cu un rang:*

$$AC_{0:7} \leftarrow AC_{1:7},\ AC_0$$

RDR; *conținutul acumulatorului se rotește spre dreapta cu un rang:*

$$AC_{0:7} \leftarrow AC_7,\ AC_{0;6}$$

Instrucțiunile de intrare/ieșire vor specifica în partea de adresă numerele (adresele) unor registre asociate interfețelor prin care sînt cuplate la unitățile de intrare/ieșire echipamentele periferice. Aceste registre, care se mai numesc și porturi de intrare/ieșire, sînt utilizate pentru a vehicula trei tipuri de informații cu interfața perifericului în cauză: date, comenzi și stări.

Datele sînt emise/recepționate de unitatea centrală/interfață periferic, pentru operațiile de ieșire și recepționate/emise de unitatea centrală/interfață periferic, în cazul operațiilor de intrare.

Comenzile sînt întotdeauna emise de unitatea centrală (sub forma unor cuvinte de comandă, în care fiecare bit are o anumită semnificație pentru periferic) și sînt interpretate — executate de interfață.

Stările sînt emise de interfața perifericului (specifică condițiile în care se află perifericul) și sînt citite de unitatea centrală în vederea luării unor decizii.

Porturile de intrare și de ieșire se vor considera organizate sub forma a două memorii notate cu *PI* și, respectiv — *PE*.

În aceste condiții se pot descrie instrucțiunile de intrare și de ieșire:

INT *ADR*16; *conținutul portului de intrare cu adresa ADR16 se transferă în acumulator:*

$$AC \leftarrow PI^{\perp ADR16}$$

IES	*ADR8*;	*conținutul acumulatorului se transferă în portul de ieșire cu adresa ADR8:* $PE \perp^{ADR8} \leftarrow AC$

Instrucţiunile din cea de-a doua clasă testează îndeplinirea unor condiţii specificate prin indicatorii de condiţii. În cazul îndeplinirii condiţiei specificate se trece la instrucţiunea a cărei adresă este dată în cîmpul de adresă, din instrucţiune, în caz contrar, se trece la instrucţiunea următoare din program (prin incrementarea contorului programului: $CP \leftarrow CP+1$).

În cele ce urmează se prezintă exemple de instrucţiuni de transfer al comenzii în program:

Cod de operație	*Adresă*	*Observații*
TPL	*ADR1*;	*testează condiția de rezultat pozitiv (plus) și se trece, în caz afirmativ, la instrucțiunea cu adresa ADR1; în caz contrar se trece la instrucțiunea cu adresa dată de* $CP \leftarrow CP+1$.
TMI	*ADR2*;	*testează condiția de rezultat negativ.*
TZE	*ADR3*;	*testează condiția de rezultat zero.*
TTR	*ADR4*;	*testează condiția de apariție a transportului.*
TNC	*ADR5*;	*transfer necondiționat al execuției programului la instrucțiunea cu adresa ADR5.*

Implementarea (în sensul de realizare) unei instrucţiuni se efectuează în două etape: citire şi execuţie.

Etapa de citire este comună pentru toate tipurile de instrucţiuni, în timp ce etapa de execuţie diferă de la instrucţiune la instrucţiune.

Etapa de citire a instrucţiunii constă în extragerea instrucţiunii din memorie, pe baza conţinutului contorului programului, şi aducerea ei în registrul instrucţiunii:

1. $RA \leftarrow CP$
2. $RD \leftarrow M \perp^{CP}$
3. $RI \leftarrow RD$

În realitate această etapă poate fi mai complexă în sensul efectuării unor operaţii asupra părţii de adresă din *RI*, în conformitate cu indicaţiile cuprinse în alte cîmpuri ale instrucţiunii (indexare, adunare cu conţinutul unui registru bază, adresare indirectă etc).

Faza de execuţie pentru instrucţiunile de prelucrare implică: aducerea operandului din memorie în registrul de date *RD*, efectuarea operaţiei specificate cu conţinutul acumulatorului şi plasarea rezultatului în acumulator):

4. $RA \leftarrow RI$ [*ADRESA*]; *partea de adresă din RI* trece în *RA*
5. $RD \leftarrow M \perp^{RA}$
6. $AC \leftarrow AC \varrho RD$
7. $CP \leftarrow CP+1$

Instrucţiunile de test şi control al secvenţei programului afectează conţinutul contorului programului după cum urmează:

5. $CP \leftarrow RI$ [*ADRESA*], în cazul condiţiei îndeplinite sau

5'. $CP \leftarrow CP+1$, în cazul în care nu se îndeplineşte condiţia testată.

5.4. Software, programare, algoritmi

După cum s-a arătat anterior, un sistem de calcul constă din echipamentele fizice de introducere/extragere, stocare şi prelucrare a informaţiei (hardware), pe de-o parte, şi din pachete de programe de sistem şi aplicaţii (software), pe de altă parte.

Sub forma cea mai generală software-ul cuprinde totalitatea programelor şi a tehnicilor de programare ale unui calculator.

Programarea are ca scop specificarea unei activităţi de prelucrare a informaţiei, într-un limbaj adecvat, în vederea efectuării ei automate, de către calculator.

Sarcina unui programator, în sensul cel mai larg, constă în descrierea sub forma unui algoritm a procesului de rezolvare a problemei şi transcrierea algoritmului într-un program, cu ajutorul unui limbaj de programare. Adesea etapa elaborării algoritmului de rezolvare a unei probleme revine unui specialist de înaltă calificare, numit analist.

Algoritmul reprezintă după cum s-a mai arătat un ansamblu de reguli sau instrucţiuni, cu următoarele caracteristici:

— este finit şi se termină după un număr finit de operaţii;

— este specificat precis, fără ambiguităţi;

— dacă necesită date de intrare, ele trebuie specificate din punctul de vedere al tipului de aplicaţii (numere întregi, reale, caractere alfanumerice etc);

— conduce întotdeauna la un rezultat (date de ieşire);

— este eficient: toate operaţiile se efectuează exact, într-un timp finit.

Pentru a rezolva o problemă dată, cu ajutorul unui calculator, se parcurg mai multe etape: analiza problemei, rafinarea algoritmului, programarea, execuţia programului, interpretarea rezultatului etc.

Analiza constă în definirea unui enunţ precis pentru problemă, precizarea datelor de intrare şi a celor de ieşire, elaborarea algoritmului de rezolvare.

Rafinarea algoritmului se referă la structurarea şi transformarea lui, în vederea obţinerii unor programe simple, bine structurate, uşor de înţeles şi bine documentate.

Se poate menţiona că o problemă dată nu conduce la un algoritm unic. O problemă este decidabilă, dacă există cel puţin un algoritm care să conducă la soluţie. În cazul cînd nu există un asemenea algoritm, problema este nedecidabilă. De asemenea, existenţa uneia sau a mai multor soluţii, pentru diversele cazuri particulare ale problemei, nu implică decidabilitatea acesteia. Invers, o problemă care nu este decidabilă, nu implică absenţa unor soluţii, ci faptul că nu există o metodă care să permită obţinerea soluţiilor, în toate cazurile, cînd ele există. În principiu nu există o metodă pentru descoperirea algoritmilor privind problemele încă nesoluţionate. Descoperirea unui algoritm reprezintă un act de creaţie, bazat pe inteligenţă, intuiţie şi, într-o bună măsură, pe experienţă.

Programarea se referă la descrierea algoritmului cu ajutorul unui limbaj de programare, care conţine instrucţiuni cu semnificaţii bine precizate pentru calculator, avînd ca rezultat generarea programului sursă.

Întrucît, la nivelul calculatorului, instrucţiunile şi datele se reprezintă sub forma unor şiruri de cifre binare, este necesară utilizarea unor notaţii specifice, care să permită scrierea programului într-un limbaj simbolic.

Instrucţiunile simbolice sînt compuse, în general, din parametri de identificare, de specificare a operaţiilor sau acţiunilor care trebuie efectuate şi de indicare a operanzilor.

Limbajele de programare s-au dezvoltat în două direcţii: limbajele de asamblare, apropiate de limbajul calculatorului, în sensul că ele folosesc notaţii simbolice (mnemotehnice), pentru a codifica instrucţiunile în cod maşină, şi limbajele evoluate, independente de calculatorul pe care se execută programul scris într-un asemenea limbaj.

Execuţia programului necesită, mai întîi, o operaţie de traducere a programului scris în limbaj de asamblare sau limbaj de nivel înalt, numit program sursă, într-un program reprezentat în instrucţiuni „cod maşina", numit program obiect.

Există două principii de traducere (translatare): traducerea globală sau compilarea şi traducerea instantanee în momentul execuţiei sau interpretarea.

Operaţia de compilare este executată de un program (compilator), aflat în memoria calculatorului, (*fig.* 5.11) pentru care programul sursă constituie datele de intrare în vederea generării datelor de ieşire, reprezentînd programul obiect.



Fig. 5.11. Compilarea şi execuţia unui program

În timpul compilării se detectează erorile sintactice, privind respectarea regulilor de scriere corectă a programului în limbajul dat, şi unele din erorile semantice, referitoare la semnificaţia programelor corecte, la erorile logice detectabile, la compilare. Problema corectitudinii programelor constituie şi în prezent un subiect de cercetare.

Prezenţa compilatorului în memoria calculatorului nu mai este necesară, după compilarea programului sursă. Programul obiect poate fi executat cu diverse seturi de date de intrare. El poate fi stocat pe un suport magnetic şi încărcat, în vederea execuţiei, ori de cîte ori este nevoie.

Interpretarea constă în traducerea programului, scris în limbaj evoluat, instrucţiune cu instrucţiune, în momentul execuţiei sale. Programul interpretor este rezident în memoria calculatorului, pe durata execuţiei programului sursă (fig. 5.12). În acest caz nu există program obiect. Limbajele interpretate / interpretative sînt, în general, interactive. Acesta este cazul limbajului BASIC, care permite dezvoltarea şi modificarea programelor într-o manieră conversaţională, facilitînd astfel punerea la punct a programelor. Limbajele interpretative conduc la programe cu o viteză mai mică de execuţie, mai puţin structurate, mai dificil de înţeles şi modificat, sub aspect logic.



Fig. 5.12. Interpretare-execuţie

Programarea în limbaje evoluate implica existenţa altor programe: utilitare, biblioteci de subrutine (pentru I/E, funcţii matematice: rădăcină pătrată, logaritmi, exponenţiale etc), sistemul de exploatare-operare.

Sistemul de exploatare (operare) permite manipularea automată a echipamentelor periferice, secvenţierea automată a unor faze ca: citirea programului sursă, compilarea/interpretarea, tipărirea la imprimantă a unor mesaje, vizualizarea rezultatelor pe display, execuţia programului obiect, editarea rezultatelor etc.

Între programele utilitare se întîlneşte şi editorul, care permite scrierea şi corectarea unui program sub forma unui text. De asemenea, se pot menţiona *programele utilitare pentru depanarea programelor sursă, a programelor pentru gestiunea fişierelor*, în cazul existenţei unor memorii externe pe disc sau bandă magnetică.

În cazul calculatorului HC-85, interpretorul pentru limbajul BASIC înglobează facilităţile de editare, de lucru cu echipamentele periferice şi subrutinele matematice, astfel încît *acest calculator apare, pentru utilizator, ca o „maşină BASIC"*.

Capitolul 6

Structura şi componentele microcalculatorului HC-85.

6.1. Configuraţie

Microcalculatorul HC-85 face parte din categoria calculatoarelor personale, neprofesionale, de uz general.

Performanţele sale superioare, costul redus, fiabilitatea ridicată, portabilitatea, extensibilitatea, disponibilitatea unui softaware de sistem şi aplicaţii, orientat către utilizator, reprezintă cîteva din caracteristicile acestui calculator personal, care recomandă utilizarea lui în diverse domenii: cercetarea ştiinţifică, proiectare asistată de calculator, învăţămînt, medicină, automatizarea unor procese industriale, controlul staţiilor pilot, al sistemelor de măsurare etc.

Într-o configuraţie completă (fig. 6.1) microcalculatorul HC-85 (se recomandă folosirea casetei de prezentare) constă din:

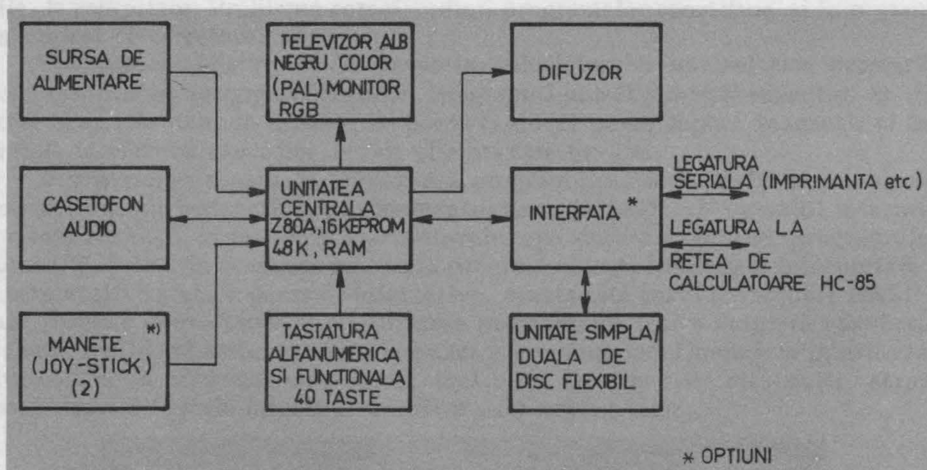


Fig. 6.1. Schema bloc a calculatorului HC-85

— unitatea centrală, pe o singură plachetă, construită cu microprocesorul Z80A, memorii EPROM — 16 Ko și RAM — 48 Ko, circuite integrate pe scară simplă și medie, pentru realizarea interfețelor cu echipamentele de I/E;
— tastatură normală, cu 40 taste, cu semnificații alfabetice și funcționale;
— televizor alb/negru sau color (PAL), monitor color RGB, pentru afișare;
— casetofon audio.

Tastatura și unitatea centrală sînt plasate în aceeași casetă (fig. 6.2)*, care este prevăzută în partea posterioară cu conectori pentru: sursa de alimentare, TV, monitor RGB, casetofon, două manete pentru jocuri (joy-stick), conector pentru interfață.



Fig. 6.2.

Aceasta din urmă asigură extinderea capabilităților sistemului, prin posibilitatea de folosire a unei unități simple/duale de disc flexibil (5" 1/4), a unei imprimate seriale de I/E și conectarea într-o rețea de calculatoare personale HC-85.

În eventualitatea că utilizatorul dorește să cupleze la HC-85 echipamente nestandard, se poate folosi conectorul plachetei unității centrale, la care sînt prezente semnalele necesare de date, adrese și comenzi. Descrierea acestor semnale se va face ulterior.

6.2. Unitatea centrală: microprocesorul Z 80, memoriile RAM, ROM

Unitatea centrală este prezentată sub formă de schemă bloc în figura 6.3. În componența ei intră următoarele resurse funcționale: unitatea centrala de prelucrare (microprocesorul Z80A), memoria ROM (16 Ko), memoria video și de date (RAM — 16 Ko), memoria suplimentară RAM (32 Ko), sincrogeneratorul, blocul de control, interfața video, interfața cu tastatura — caseta magnetică și difuzorul, codificatorul PAL, modulatorul TV și sursa de alimentare.

Unitatea centrală de prelucrare folosește *microprocesorul Z80A*, pe 8 biți, realizat în tehnologia NMOS și plasat într-o capsulă cu 40 terminale.

Terminalele, în afara celor care asigură tensiunea de alimentare +5V, masa, ceasul și inițializarea (RESET), se pot considera conectate la magistralele de date, adrese și comenzi.

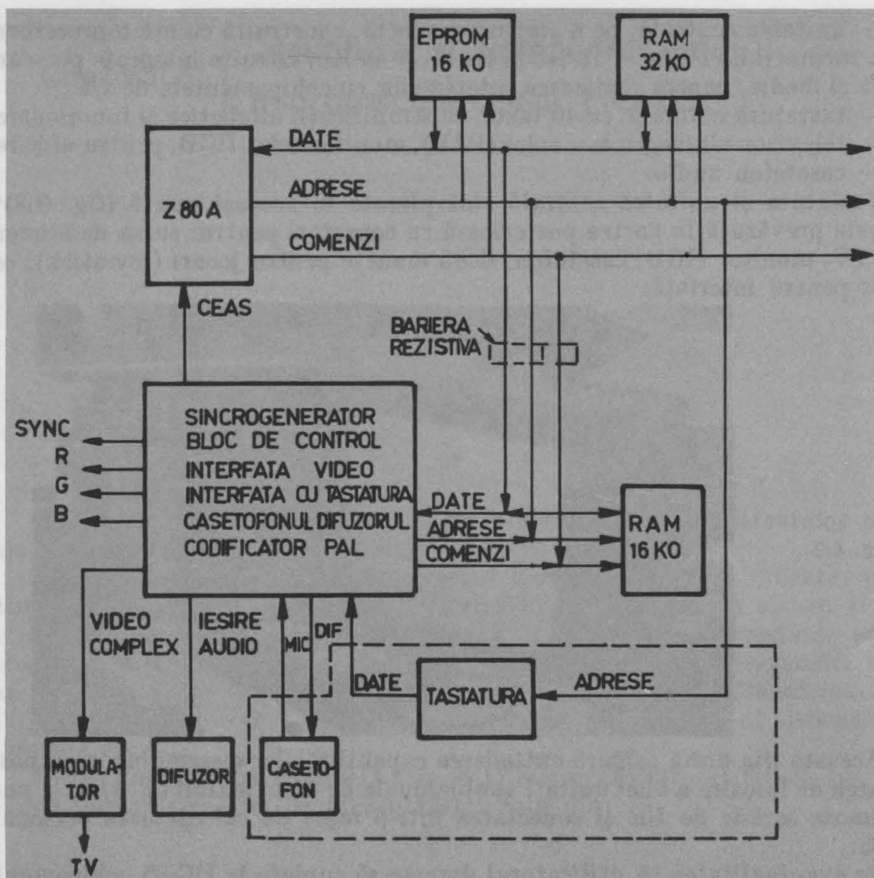


Fig. 6.3. Schema bloc a unității centrale, la nivel funcțional

Magistrala de date (D0—D7) este bidirecțională și poate intra în starea de mare impedanță. Ea este folosită pentru schimbul de informații între procesor, pe de-o parte și memoria/dispozitivele de I/E, pe de altă parte.

Magistrala de adrese (A0—A15) furnizează adresele necesare pentru selecția celulelor de memorie și a porturilor de I/E. Spațiul de adresare permite manipularea a 65536 celule de memorie și respectiv, a 256 porturi de I/E.

Magistrala de comenzi cuprinde semnalele necesare coordonării transferului de date între microprocesor și memorie/porturi de I/E.

Pe lîngă alte operații, asociate cu întreruperile și cererile de magistrală, microprocesorul execută în principal următoarele activități:

— citire/scriere date, din/in memorie,
— citire/scriere date, de la/la un port de I/E,
— operații aritmetice-logice asupra datelor.

Microprocesorul Z80A executa 158 de instrucțiuni distincte. În cazul de față, el operează la o frecvență de tact de 3,5 MHz.

În figura 6.4 se prezintă terminalele microprocesorului, care au următoarele semnificații:

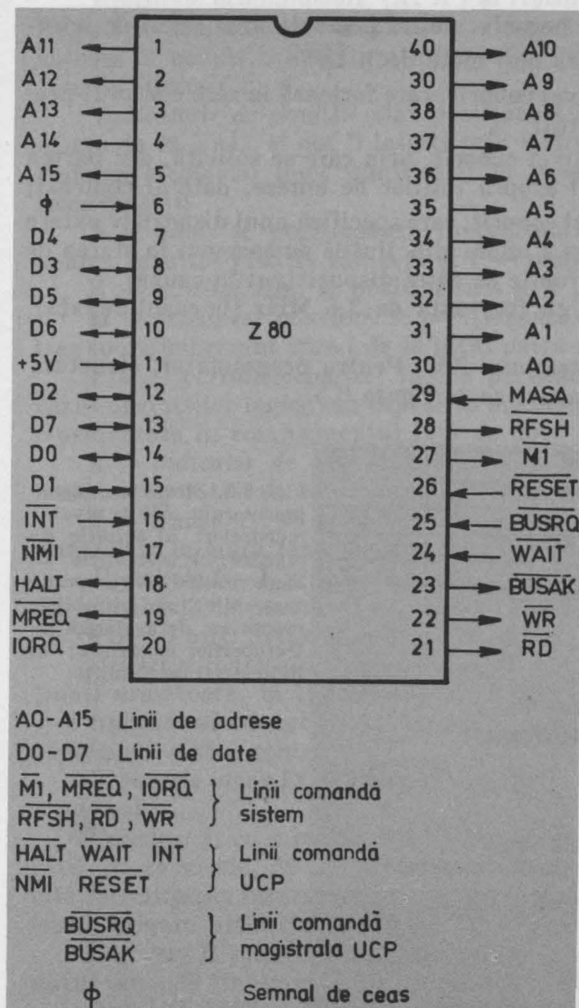


Fig. 6.4. Terminalele microprocesorului Z80

A0 — A15: liniile semnalelor de adrese reprezintă ieşiri cu trei stări, active pe nivel ridicat. Liniile **A0—A7** sînt folosite pentru a selecta unul din cele 256 porturi de I/E. Pe durata perioadei de reîmprospătarea memoriei dinamice, biţii **A0—A6** conţin adresa de reîmprospătare;

D0 —D7: liniile semnalelor de date au un caracter bidirecţional, sînt active pe nivel ridicat, pot intra în starea de mare impedanţă şi asigură transferul informaţiei între microprocesor şi memorie / porturile de I/E;

$\overline{\text{M1}}$: linie de ieşire, activă pe nivel coborît, indică ciclul maşină în care se citeşte primul octet al unei instrucţiuni;

$\overline{\text{MREQ}}$: linie de ieșire cu trei stări, activă pe nivel coborît, specifică prezenţa unei adrese de celulă de memorie, pe liniile **A0 —A15**;

$\overline{\text{IOREQ}}$: linie de ieșire cu trei stări, activă pe nivel coborît, indică prezenţa unei adrese de port de I/E, pe liniilie **A0 —A7**;

$\overline{\text{RD}}$: linie de ieşire cu trei stări, activă pe nivel coborît, specifică o operaţie de citire de date, de la memorie sau de la un port de I/E, care trebuie să forţeze datele pe liniile **D0 —D7**;

$\overline{\text{WR}}$: ieșire cu trei stări, activă pe nivel coborît, indică prezenţa datelor pe liniile **D0 —D7**, pentru a fi înscrise în memorie sau la un port de ieșire;

$\overline{\text{RFSH}}$: ieșire, activă pe nivel coborît, specifică prezenţa, pe liniile **A0 —A7**, a adresei de reîmprospătare pentru memoria dinamică;

$\overline{\text{HALT}}$: ieşire, activă pe nivel coborît, indică terminarea execuţiei instrucţiunii **HALT**;

$\overline{\text{WAIT}}$: intrare, activă pe nivel coborît, prin care memoria sau portul adresate semnalizează că nu sînt pregătite pentru transferul de date;

$\overline{\text{INT}}$: intrare, activă pe nivel coborît, reprezintă o cerere de întrerupere din partea unui echipament de I/E;

$\overline{NMI}$: intrare, activă pe front negativ, indică prezenţa unei cereri de întrerupere nemascabile, cu prioritatea mai mare decît $\overline{INT}$;

$\overline{RESET}$: intrare, activă pe nivel coborît, care forţează în zero contorul programului şi iniţializează procesorul;

$\overline{BUSRQ}$: intrare, activă pe nivel coborît, prin care se solicită, din partea unui dispozitiv extern, controlul asupra liniilor de adrese, date şi comenzi;

$\overline{BUSAK}$: ieşire, activă pe nivel coborît, care specifică unui dispozitiv extern trecerea liniilor de adrese, date şi a unora din liniile de comenzi în starea de mare impedanţă, pentru a fi controlate de către dispozitivul în cauză;

ϕ: semnal de ceas monofazic, cu frecvenţa de 3,5 MHz (în cazul de faţă), generat extern.

Structura internă a microprocesorului Z80. Pentru programator, structura internă a microprocesorului Z80 apare ca în figura 6.5.

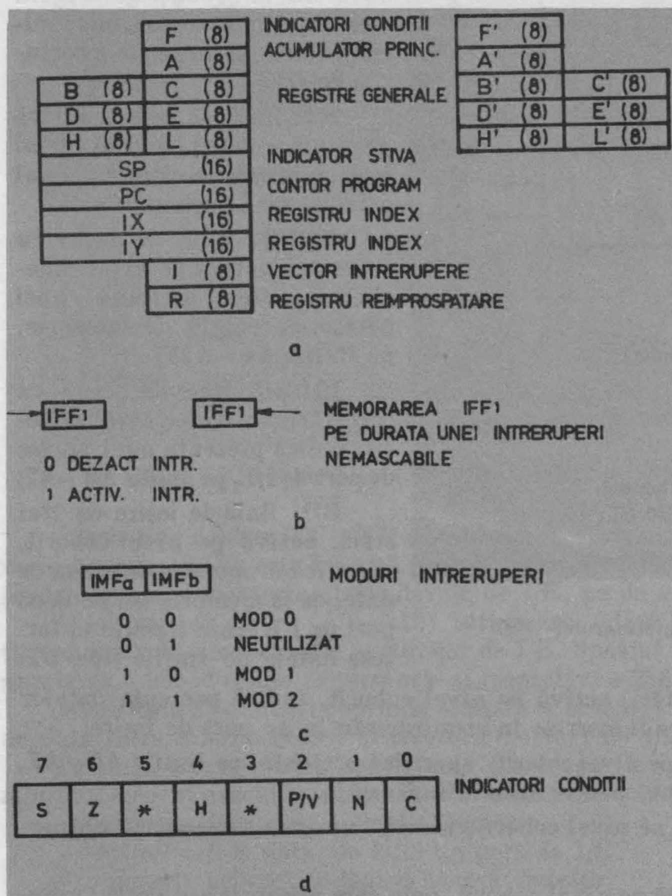


Fig. 6.5. Structura microprocesorului Z80 la nivelul registrelor: a) seturile de registre; b) bistabilele de stare pentru întreruperile mascabile; c) bistabilele modurilor de tratare a întreruperilor mascabile; d) indicatorii de condiţii

Microprocesorul Z80 dispune de două seturi de registre, cel de-al doilea fiind alcătuit din dublurile registrelor **F**, **A**, **B**, **C**, **D**, **E**, **H**, **L**. Trecerea de la un set la altul se efectuează prin instrucţiunea EXCHANGE (EX AF, AF').

Registrele acumulatoare (**A, A'**) *și registrele indicatoare de condiții* (**F, F'**). Registrele acumulatoare sînt organizate pe 8 biţi şi au asociate registrele indicatoare de condiţii. Structura informaţiei şi registrele de condiţii sînt prezentate în figura 6.5. d.

Indicatorii de condiţii sînt poziţionaţi automat, ca urmare a operaţiilor efectuate în UAL, și pot fi testaţi prin instrucţiuni de transfer condiţionat, în vederea efectuării unor ramificaţii în program. Indicatorii au următoarele semnificaţii:

S — *semn*: se poziţionează în conformitate cu semnul rezultatului (0 — pentru rezultat pozitiv sau zero şi 1 — pentru rezultat negativ);

Z — *rezultat zero*: se poziţionează în 1 pentru un rezultat egal cu zero;

H — *transport auxiliar*: se poziţionează în 1 ca urmare a apariţiei unui transport/împrumut spre / de la bitul patru al acumulatorului;

P/V — *paritate/depăşire*: indică paritatea rezultatului în acumulator, în cazul operaţiilor logice sau depăşirea aritmetică, în cazul operaţiilor cu numere reprezentate în complementul faţă de doi;

N — *indicator de adunare/scădere*: specifică tipul instrucţiunii executate înaintea operaţiei de corecţie, la operarea în binar-zecimal;

C — transport: se poziţionează în 1 ca urmare a apariţiei unui transport/ împrumut în afara rangului de semn / în rangul de semn.

Registrele **B—L** şi **B'—L'** pot fi folosite individual, ca registre de 8 biţi, sau asamblate în perechi **B—C, D—E, H—L** şi **B'—C', D'—E', H'—L'**, ca registre de 16 biţi. Seturile de registre se pot selecta prin instrucţiunea **EXX**.

Registrul contorului programului **PC** are 16 biţi şi conţine adresa instrucţiunii următoare, în timpul execuţiei instrucţiunii curente.

Indicatorul adresei vîrfului stivei **SP** are 16 biţi şi indică adresa celulei care reprezintă vîrful stivei.

Registrele index **IX** şi **IY** au cîte 16 biţi; ele conţin constantele de indexare a adresei.

Registrul **I**, cu o lungime de 8 biţi, permite adresarea indirectă a unei locaţii de memorie, în urma unei cereri de întrerupere; perifericul care solicită întreruperea furnizează primii opt biţi mai semnificativi, în timp ce registrul **I** asigura ultimii opt biţi mai puţin semnificativi.

Registrul **R** este folosit pentru reîmprospătarea memoriei dinamice. Conţinutul său este transmis pe liniile **A0—A6**, simultan cu semnalul **RFSH**.

Bistabilii **IFF1** şi **IFF2** specifică starea sistemului de întrerupere, al microprocesorului, pentru întreruperile mascabile. **IFF1=0/1** — dezactivat/activat; **IFF2=IFF1** pe durata servirii unei întreruperi nemascabile.

Bistabilii **IMFa, IMFb** specifică modurile programate, pentru răspunsul la cererile de întrerupere mascabile: *modul* 0—**IMFa/IMFb**=0/0, *modul* 1—**IMFa/IMFb**=1/0, *modul* 2—**IMFa/IMFb=1/1**

Memoria adresată de microprocesor are o capacitate totală de 64 *Ko, din care* 16 *Ko — cu conţinut fix (EPROM) şi* 48 *Ko — cu conţinut variabil (RAM).*

Memoria EPROM, care ocupă spaţiul de adresare de la 0 la 16383, conţine codurile maşină ale interpretorului, pentru limbajul BASIC, şi subrutinelor folosite pentru manipularea hardware-lui microcalculatorului HC-85 (subrutina care permite citirea caracterelor introduse de la tastatură, subrutina care controlează sunetul la difuzor, subrutina pentru scrierea/citirea de la/la casetofon etc). Această memorie este realizată fizic cu circuitele 2716, avînd fiecare o capacitate de 2 Ko.

Memoria EPROM este conectată direct pe magistrala UCP şi operează independent de restul memoriei. Ea poate fi dezactivată forţînd semnalul ROMCS, de la conectorul cu 28 de terminale al plachetei, la +5V. În acest mod, la conector, se poate plasa o altă memorie EPROM, cu alt conţinut.

Memoria RAM ocupă spaţiul de adresare de la 16384, la 65535. Ea este împărţită în două secţiuni.

Prima secţiune, cu ocapacitate de* 16 *Ko, plasată în spaţiul de adresare* 16384 — 32767, *poartă numele de memorie video şi de program. Ea stochează, atît informaţia care se afişează pe ecranul televizorului, cît şi informaţia suplimentară, care se referă la: atributele caracterelor afişate pe ecran, tamponul datelor pentru imprimantă (32 de caractere — o linie ecran), variabilele de sistem, programul BASIC etc.

Memoria video şi de program este citită la intervale fixe de către sincrogenerator, pentru a transmite spre TV informaţia video şi atributele de culoare.

Unitatea centrală de prelucrare adresează, de asemenea, memoria video şi de program, în scopul modificării imaginii, a atributelor de culoare, a variabilelor de sistem sau pentru a stoca programe BASIC sau date.

Din cele de mai sus rezultă că această memorie este de tip biport, impunîndu-se rezolvarea conflictelor, care apar la coincidenţa cererilor de acces din partea sincrogeneratorului şi a UCP. Conflictul se va rezolva întotdeauna în favoarea sincrogeneratorului, prin oprirea temporară a ceasului UCP. Aceasta situaţie provoacă unele neajunsuri în privinţa programelor care au bucle de temporizare şi care sînt stocate în această secţiune de memorie. Pentru controlul exact al temporizărilor, prin program, se recomandă plasarea programelor respective în memoria suplimentară.

Sincrogeneratorul adresează memoria video pentru a citi doi octeţi: unul din zona de afişare, în scopul controlării a 8 puncte de pe ecran şi altul din zona de atribute, pentru a comanda culoarea.

A doua secţiune a memoriei RAM reprezintă memoria suplimentară, cu o capacitate de 32 *Ko, plasată în spaţiul de adresare* 32768—65535. Ea este folosită pentru stocarea programelor BASIC, a datelor etc.

Memoria suplimentară este legată direct pe magistrala de date a UCP şi prin intermediul unor multiplexoare, la magistrala de adrese. Aceasta permite

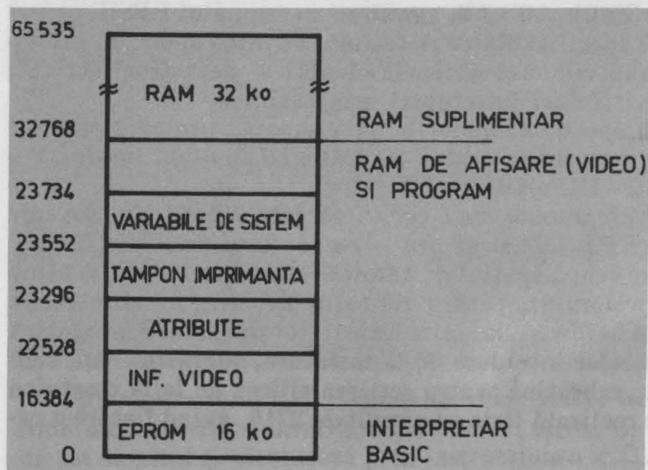


Fig. 6.6. Structura memoriei după conţinut

utilizarea ei de către UCP, chiar dacă sincrogeneratorul adresează memoria video, deoarece magistralele lor sînt separate printr-o barieră rezistivă.

Memoria RAM este realizată cu circuite dinamice 4116, cu o capacitate de 16 Ko/circuit.

Organizarea informației în memorie este prezentată în figura 6.6.

6.3. Subsistemele de intrări-ieşiri: interfețe cu televizorul, tastatura, casetofonul, difuzorul; conectorul de extensii

Sincrogeneratorul asigură semnalele de adresă şi comandă pentru explorarea periodică a memoriei video, semnalele primare de sincronizare de linie şi cadre, în vederea obținerii unei imagini stabile pe ecranul TV.

Imaginea pe ecran este reprezentată prin 256 puncte, pe fiecare din cele 192 linii utile pe cadru. Astfel, se pot afişa pe ecran 24 de rînduri, cu cîte 32 de caractere pe rînd.

Atributele de culoare sînt stabilite la nivel de caracter (matrice de 8×8). În zona de 8×8 puncte a unui caracter, un punct poate avea numai două culori, la un moment dat: „cerneala" (INK) şi „hîrtia" (PAPER), specificate cu cîte 3 biți, în octetul de atribute. *Formatul octetului unui atribut, pentru un caracter (8×8) puncte este următorul:*

D7	D6	D5 D4 D3	D2 D1 D0
F	*	PAPER	INK

F — afişare intermitentă a caracterului (flash-clipitor), activ pe 1 (D7);
PAPER — specifică una din cele 8 culori pentru fond (**D5—D3**)
INK — indică una din cele 8 culori pentru punctele caracterului (**D0—D2**).

Codurile culorilor sînt următoarele:

000 — negru	100 — verde
001 — albastru	101 — bleu (cyan)
010 — roşu	110 — galben
011 — magenta	111 — alb.

După cum s-a mai arătat, informația privind imaginea TV este stocată în memorie, în două zone adiacente:

— *zona de informații video*, cu adresele 16384—22527,

— *zona de atribute video*, cu adresele: 22528—23295.

Zona de informații specifică pentru fiecare punct dacă este de tip „PAPER" (0) sau de tip „INK" (1).

Zona de atribute va indica, pentru fiecare matrice de 8×8 puncte culorile.

Blocul de control asigură semnalele de selecție pentru memoria RAM video şi program.

Interfața video generează semnalele standard R, G, B şi de sincronizare, pentru monitorul color, utilizînd informația din memoria video. La fiecare grup de 8 octeți, reprezentînd un caracter, corespunde un octet de atribute, cu structura dată mai sus. Aceste informații sînt înscrise în mai multe registre, dintre care unele funcționează serial, pentru a asigura controlul imaginii la nivel de punct.

Interfaţa video mai furnizează informaţia de culoare pentru bordura ecranului, folosind biţii **D0—D2**, ai portului de ieşire cu adresa 254 (FE). De asemenea, generează informaţia de sincronizare linie/cadru/stingere, pe baza semnalelor primare furnizate de sincrogenerator.

Interfaţa cu tastatura asigură preluarea informaţiei de la cele 40 de taste cu semnificaţii alfanumerice şi funcţionale.

Tastatura este formată dintr-o matrice de 8×5 trasee. La intersecţiile liniilor şi coloanelor sînt plasate tastele. Liniile matricei sînt conectate la adresele **A8—A15**, ale magistralei de adrese. Coloanele matricei sînt legate la liniile **D0—D4**, ale magistralei de date. Conexiunile respective sînt efectuate prin intermediul unor circuite separatoare. Tastatura este explorată la fiecare 20 ms, prin lansarea unei cereri de întrerupere, către UCP, la sfîrşitul afişării fiecărui cadru video. Ca urmare a întreruperii, UCP va executa instrucţiuni de intrare (IN) de la portul 254 (FE). Traseele coloanelor sînt conectate la +5 V prin intermediul unor rezistenţe de 10 KΩ (fig. 6.7). Pentru a stabili tasta apăsată, care va realiza un contact galvanic la intersecţia coloanei cu linia respectivă, este necesar ca, periodic, una din liniile de adrese să fie adusă la un potenţial coborît (0 logic), în timp ce liniile celelalte de adrese sînt menţinute la un potenţial ridicat (1 logic). Bitul poziţionat în 0, în grupul **D0—D4**, va corespunde coloanei conectate la linia de adrese forţată la zero.

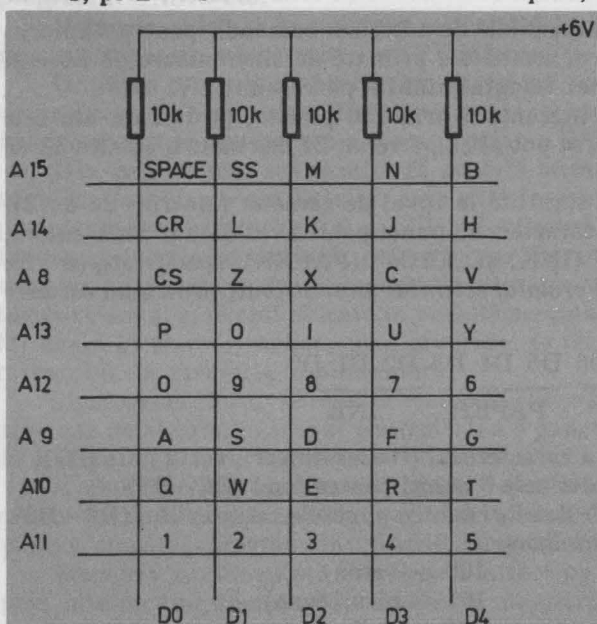


Fig. 6.7. Matricea de trasee a tastaturii

În scopul explorării, în maniera arătată, a liniilor de adresă **A15,—A8** menţinînd pentru **A7—A0** valoarea 254, este necesar să se execute instrucţiuni de intrare cu următoarele adrese:

Instrucţiuni	*Linie de adrese forţată la zero*	*Taste selectate.*
IN 32766	**A15**	**SPACE, SS, M, N, B**
IN 49150	**A14**	**CR, L, K, J, H**
IN 57324	**A13**	**P, O, I, U, Y**
IN 61438	**A12**	**0, 9, 8, 7, 6**
IN 63486	**A11**	**1, 2, 3, 4, 5**
IN 64510	**A10**	**Q, W, E, R, T**
IN 65022	**A9**	**A, S, D, F, G**
IN 65278	**A8**	**CS, Z, X, C, V**

Pentru detalii privind specificaţiile tastelor se poate apela la caseta de prezentare sau/şi la paragraful corespunzător din capitolul referitor la limbajul BASIC.

Interfaţa cu casetofonul asigură citirea/stocarea programelor şi a datelor de pe/pe caseta magnetică.

Conectorul audio are la terminale următoarele semnale:

1,4 — ieşire: nivel 500mV, impedanţa de ieşire 500Ω.

3,5 — intrare: 1—4V, impedanţa sursei de semnal max. 10 KΩ.

Ieşirea audio este comandată cu ajutorul unui bistabil a cărui stare este controlată prin bitul **D3**, al portului de ieşire, cu adresa 254 (FE). Forţînd alternativ în 0 sau 1 acest bit, se poate genera o formă de undă dreptunghiulară la ieşirea liniei audio. Cu un volum constant, frecvenţa semnalului va depinde de durata intervalului de timp în care starea bistabilului de ieşire a rămas neschimbată.

Citirea informaţiei de pe suportul magnetic implică amplificarea şi filtrarea semnalului, ceea ce va permite, în continuare, comanda bitului **D6**, al portului de intrare cu adresa **254 (FE)**.

Pe suportul magnetic, fişierele sînt înregistrate sub forma a două blocuri numite *header* (*antet*) şi *bloc de date*.

Header-ul constă din 19 octeţi de date, dintre care numai 17 sînt generaţi de utilizator (fig. 6.8).

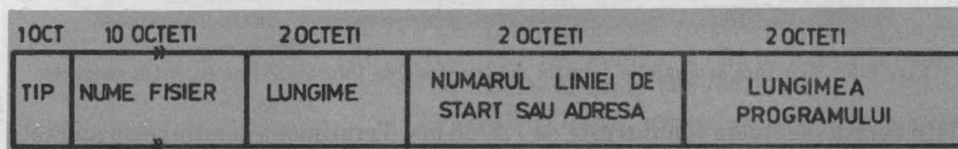


Fig. 6.8. Structura blocului header

Primul octet al *header*-ului sau al *blocului de date*, *notat cu* **FLAG**, este generat de rutina de salvare SAVE, pentru a face deosebirea între un bloc *header* (FALG=0) şi un *bloc de date* (FLAG=255).

Octetul 19 al *header*-ului sau *ultimul octet al blocului de date* conţine informaţia de paritate, ceea ce permite detectarea erorilor la încărcarea informaţiilor de pe casetă.

Primul octet şi ultimul octet sînt adăugaţi în mod automat de către rutina SAVE.

Cei 17 octeţi ai blocului *header*, furnizaţi de utilizator, au următoarele semnificaţii:

TIP — un octet, tipul *blocului de date:*

0 — program BASIC,
1 — tablou numeric,
2 — tablou de tip şir,
3 — cod maşină sau imagine ecran.

NUME FIŞIER — zece octeţi. **LUNGIMEA** *blocului de date* — doi octeţi.

Semnificaţiile următorilor patru octeţi depind de tipul *blocului de date* descris în *header*. Dacă **TIP**=0, atunci octeţii 14 şi 15 specifică numărul liniei pentru *autostart*, al programului BASIC, iar octeţii 16 şi 17 indică numărul de

octeţi din partea de program a fişierului (salvarea programului BASIC implică salvarea zonei de program şi de date). Dacă **TIP**=1, 2, se utilizează numai octetul 15, pentru a specifica numele tabloului. Dacă **TIP**=3, se folosesc numai octeţii 14 şi 15, pentru a specifica adresa de la care trebuie să se încarce octeţii de date. *Blocul de date* este precedat de octetul **FLAG** (255) şi terminat cu octetul de paritate. Ceilalţi octeţi ai blocului reprezintă imagini ale zonei de memorie salvate. Octetul de paritate este generat înainte de salvarea pe suport magnetic a *header*-ului şi a *blocului de date*, prin aceea că fiecare octet, care este transmis spre ieşire, reprezintă rezultatul unei operaţii SAU-EXCLUSIV între informaţia propriu-zisă şi octetul curent de paritate. Valoarea iniţială a octetului de paritate este chiar valoarea octetului **FLAG**. La citirea informaţiei de pe suportul magnetic are loc o operaţie similară. Citirea corectă a blocului conduce la o valoare finală 0, a octetului de paritate.

Semnalul fizic pentru fiecare bloc începe cu un ton rafala, care durează 5 s, *pentru header şi* 2 s, *pentru blocul de date* (fig. 6.9). Perioada undei dreptunghiu-

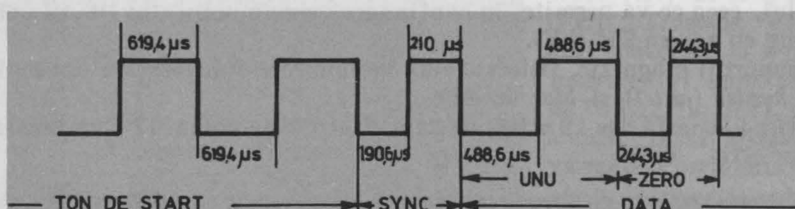


Fig. 6.9. Structura semnalelor folosite pentru stocarea informaţiei pe suportul magnetic

lare corespunzătoare tonului este de 1,2388 ms. Terminarea tonului este marcată de un semnal de sincronizare, cu o valoare coborîtă pe durata a 190,6 µs şi cu valoare ridicată pe durata a 210 µs. În continuare, fără nici o pauză, se transmit datele, care reprezintă şiruri de biţi 0 şi 1. Zero este reprezentat printr-un semnal, care are valoarea coborîtă timp de 244,3 µs şi valoarea ridicată tot timp de 244,3 µs. Unu are drept corespondent un semnal similar, dar cu durate duble pentru cele două stări.

Interfaţa pentru difuzor primeşte un semnal dreptunghiular, corespunzător modificării stării unui bistabil, comandat de bitul **D4**, al portului de ieşire cu adresa **254** (**FE**).

Codificatorul PAL are ca intrări semnalele **R**, **G**, **B** şi semnalul de sincronizare **SY**. Pe baza semnalelor **R**, **G**, **B** se obţin semnalele **Eu** şi **Ev**, reprezentînd diferenţele între semnalele **B** şi **G**, respectiv **R** şi **G**, care, împreună cu semnalul de *burst*, obţinut din **SY**, se aplică la intrările circuitului TCA 650. Acest circuit generează semnalul de crominanţă, frecvenţa subpurtatoare **PAL** obţinîndu-se de la un oscilator cu frecvenţa de 4,433618 MHz. Un alt circuit generează semnalul de luminanţă, pornind tot de la semnalele **R**, **G**, **B**. Prin combinarea într-un etaj final a semnalelor de crominanţă şi luminanţă se obţine semnalul video complex, codificat **PAL**. Acesta poate fi utilizat pentru atacarea unui monitor **PAL** sau, după modulare, a unui televizor color.

Sursa de alimentare este constituită dintr-un alimentator extern, care furnizează o tensiune redresată de minimum 9 V, în sarcină. Tensiunea de +5 V se obţine cu ajutorul unui stabilizator integrat LM 7805, plasat pe un radiator. Tensiunile de +12 V şi −5 V sînt generate de un convertor cu doi tranzistori

şi cu componentele pasive corespunzătoare. Pentru alimentarea eventualelor montaje, plasate pe conectorul de extensie, se recomandă limitarea curentului la maximum 100 mA, de la sursa de +5 V.

Conectorii. Pentru cuplarea unor echipamente periferice convenţionale / neconvenţionale, în varianta standard a calculatorului HC-85, sînt prevăzuţi mai mulţi conectori: de extensie, video, pentru manete de tip joy-stick, audio şi de antenă.

Conectorul de extensie are 2×28 terminale (fig. 6.10) la care sînt accesibile liniile magistralelor de date, adrese şi comenzi ale sistemului. Semnificaţiile semnalelor sînt identice cu cele prezentate în cadrul descrierii microprocesorului Z80.

B		A
A11	28	NC
A9	27	A10
$\overline{\text{BUSACK}}$	26	A8
$\overline{\text{ROMCS}}$	25	$\overline{\text{RFSH}}$
A4	24	$\overline{\text{M1}}$
A5	23	NC
A6	22	+12V
A7	21	$\overline{\text{WAIT}}$
$\overline{\text{RESET}}$	20	-5V
$\overline{\text{BUSRQ}}$	19	$\overline{\text{WR}}$
NC	18	$\overline{\text{RD}}$
NC	17	$\overline{\text{IORQ}}$
NC	16	$\overline{\text{MREQ}}$
NC	15	$\overline{\text{HALT}}$
GND	14	$\overline{\text{NMI}}$
$\overline{\text{IORGE}}$	13	$\overline{\text{INT}}$
A3	12	D4
A2	11	D3
A1	10	D5
A0	9	D6
CLK	8	D2
GND	7	D1
GND	6	D0
SLOT	5	SLOT
NC	4	NC
+5V	3	D7
A12	2	A13
A14	1	A15

Fig. 6.10. Conectorul de extensie

În continuare se vor face unele precizări privind particularităţile de folosire a unora dintre semnalele accesibile la conector.

13A — $\overline{\text{INT}}$: intrare, întrerupere mascabilă conectată la terminalul $\overline{\text{INT}}$ al microprocesorului şi printr-o rezistenţă de 680 Ω la logica generatoare de întreruperi, de pe placa UC. Este activă pe nivel coborît şi poate fi utilizată de un dispozitiv extern, pentru a solicita întreruperi.

14A — $\overline{\text{NMI}}$: intrare, întrerupere nemascabilă, activă pe front negativ. Poate fi utilizată de un dispozitiv extern, pentru a forţa execuţia unei rutine cu adresa de start **102 (66H)**.

17A — $\overline{\text{IORQ}}$: ieşire, semnal de comandă pentru I/E, activ pe nivel coborît. Partea mai puţin semnificativă a magistralei de adrese **A0—A7**, specifică adresa unui port de I/E. În BASIC, cu o instrucţiune de I/E se poate specifica o adresă de 16 biţi, care apare pe liniile de adrese **A0—A15**, cînd $\overline{\text{IORQ}}$ este zero.

21A — $\overline{\text{WAIT}}$: intrare, activă pe nivel coborît, este folosită de echipamentele mai lente pentru a se sincroniza cu microprocesorul. Linia $\overline{\text{WAIT}}$ nu trebuie activată mai mult de 1 ms, pentru a nu bloca reîmprospătarea memoriei dinamice.

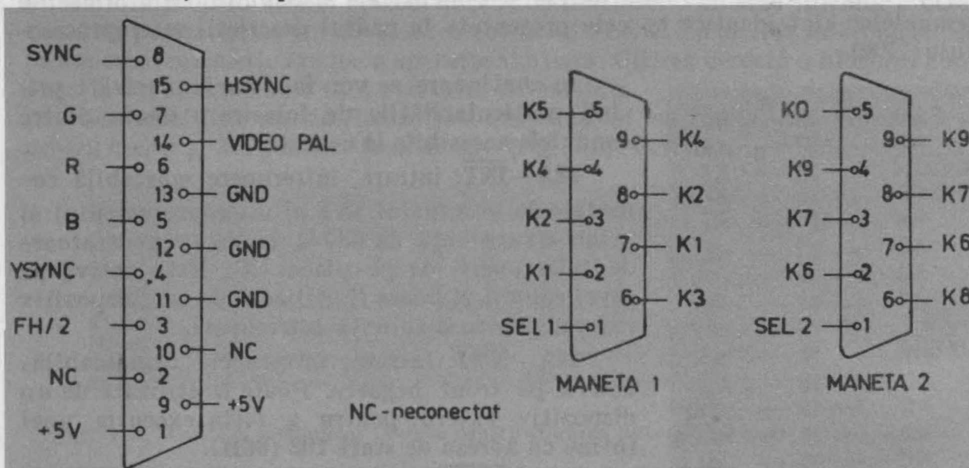
8B—CLOCK: ieşire, semnal de ceas 3,5 MHz. Poate fi utilizat de echipamente externe pentru sincronizarea cu microprocesorul; este blocat cînd procesorul solicită accesul la memoria video, în acelaşi timp cu sincrogeneratorul.

13B — $\overline{\text{IORGE}}$: semnal de intrare care, conectat la +5 V, blochează semnalul $\overline{\text{IORQ}}$, ce se aplica la placa UC.

19B — $\overline{\text{BUSRQ}}$: semnal de intrare, conectat la terminalul cu acelaşi nume al microprocesorului, prin care un echipament extern solicită preluarea controlului asupra magistralelor de date, adrese şi a unora dintre comenzi.

25B—ROMCS: semnal de intrare activ pe nivel ridicat; conectat la +5 V dezactivează memoria EPROM de pe placheta şi permite activarea unei memorii plasate pe conectorul de extensie.

Conectorul video (fig. 6.11) asigură semnalele necesare conectării unor monitoare color **R, G, B** (*Electronica* 001), **PAL** (*Electronica* 002), monitoare alb/negru, cu intrare video compus, Semnalul **FH/2** are frecvenţa egală cu jumătate din frecvenţa de linie. Împreună cu semnalele **SYNC, R, G, B** se poate folosi pentru a obţine un semnal video codificat, necesar unui alt sistem TV, utilizînd o interfaţă corespunzătoare.



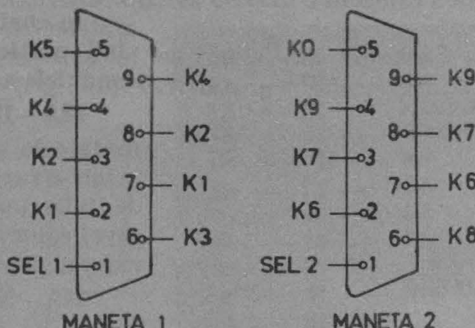


Fig. 6.11. Conectorul video

Fig. 6.12. Conectorii manetelor 1 şi 2

Conectorii pentru manete (fig. 6.12) dublează funcţiile primului rînd de taste. Manetele asigură închiderea unor contacte (de microintreruptor) normal deschise, atunci cînd sînt activate în direcţiile stînga, dreapta, sus, jos. Pe conectori sînt notate tastele dublate în momentul închiderii contactului între terminalul respectiv şi terminalele notate cu **SEL1** şi respectiv **SEL2**

Funcţii	*Maneta*1	*Maneta*2
Stînga	**Tasta1**	**Tasta6**
Dreapta	**Tasta2**	**Tasta7**
Jos	**Tasta3**	**Tasta8**
Sus	**Tasta4**	**Tasta9**
Foc	**Tasta5**	**Tasta0**

Conectorul audio serveşte la cuplarea unui casetofon audio, pentru citirea/stocarea programelor şi datelor. Terminalele sale au următoarele semnificaţii:

1,4 — ieşire 500 mV,
3,5 — intrare 1—5 V,
2 — masă.

Conectorul TV este folosit pentru cuplarea unui televizor, prin mufa de antenă.

Capitolul 7 | Elemente de programare în limbaj algoritmic

7.1. Algoritmi

În activitățile cu caracter practic se întîlnesc numeroase probleme simple sau complexe, pentru a căror rezolvare se utilizează sisteme de reguli bine definite (instrucţiuni sau comenzi), care indică executantului procedeele de operare. Sistemele de reguli pot fi studiate înainte de rezolvarea problemei sau pot fi elaborate în procesul rezolvării acesteia.

În cazul în care executantul este un automat sau un calculator, sistemul de reguli trebuie descris în limbajul comenzilor sau instrucţiunilor specifice executantului.

Astfel, algoritmul reprezintă o descriere precisă a succesiunii de activităţi pe care trebuie să le efectueze executantul, pentru atingerea unui anumit obiectiv (scop) sau pentru rezolvarea problemei date. În continuare se vor prezenta cîteva exemple de algoritmi.

Exemplul 1. *Algoritmul lui Euclid* pentru găsirea celui mai mare divizor comun a două numere naturale a şi b.

Algoritmul se bazează pe proprietatea că *dacă* $a>b$, atunci cel mai mare divizor comun, pentru a şi b, este acelaşi cu cel al numerelor $a-b$ şi b.

Se va proceda după cum urmează:

1. *dacă* $a=b$, oricare din ele reprezintă rezultatul, *în caz contrar se trece la punctul* 2;
2. se determină care din cele două numere este mai mare şi se trece la punctul următor;
3. se înlocuieşte numărul mai mare cu diferenţa dintre numărul mai mare şi numărul mai mic, se trece la punctul următor;
4. se trece la punctul 1.

Operaţiile specificate în acest algoritm se vor repeta pînă la obţinerea rezultatului.

Algoritmii pot descrie şi alte activităţi, care nu implică efectuarea unor calcule.

Exemplul 2. *Algoritmul pentru construcţia perpendicularei într-un punct dat C, al segmentului de dreaptă $\overline{AB}$, cu ajutorul compasului şi al riglei.* Se considera $|\overline{AC}| < |\overline{CB}|$.

1. cu vîrful compasului plasat în C şi cu deschiderea $|\overline{AC}|$ se găseşte pe segmentul $\overline{CB}$ un punct D;

2. cu vîrful compasului plasat în D, se trasează un cerc cu raza $r = |\overline{AD}|$;

3. cu vîrful compasului plasat în A se trasează un cerc cu raza $r = |\overline{AD}|$;

4. cu ajutorul riglei se unesc punctele de intersecţie ale celor două cercuri dreapta astfel obţinută este perpendiculară în punctul C, la segmentul $\overline{AB}$.

Execuţia comenzilor în ordinea indicată conduce la rezultatul dorit.

Exemplul 3. *Algoritmul „ghicirii" unui număr x, la care s-a gîndit partenerul de joc.*

Se propune partenerului de joc efectuarea următoarelor operaţii asupra numărului ales:

1. se înmulţeşte numărul cu 5;
2. la rezultat se adună 6;
3. suma se înmulţeşte cu 2;
4. se comunică rezultatul *r* obţinut.

Pentru a afla numărul *x*, se procedează astfel:

1. din rezultatul *r* se scade 12;
2. din rezultatul obţinut la punctul anterior, se înlătură cifra cea mai puţin semnificativă, numărul obţinut fiind chiar numărul *x*

Din ultimul exemplu se constată că executantul poate efectua automat cele două operaţii legate de determinarea numărului *x*, fără a soluţiona ecuaţia $(5 \cdot x+6) \cdot 2=r$.

Aceasta arată că algoritmul permite rezolvarea problemei fără ca executantul să înţeleagă semnificaţiile operaţiilor efectuate.

Execuţia algoritmului are un caracter formal şi poate fi efectuată mecanic, de către un automat corespunzător sau un calculator, prevăzut cu un program adecvat.

Elaborarea algoritmului, pentru rezolvarea unei probleme dintr-un domeniu dat, după cum s-a mai arătat, necesită cunoştinţe profunde, atît în domeniul respectiv, cît şi în cel al matematicilor. Astfel, după formularea problemei, se impune găsirea modelului matematic sau procedural corespunzător. În continuare se elaborează algoritmul, se face verificarea corectitudinii şi a complexităţii sale. După stabilirea algoritmului problema poate fi rezolvată în mod mecanic.

7.2. Limbajul algoritmic (pseudocod)

Una din etapele importante în pregătirea execuţiei mecanice a unui algoritm o constituie descrierea lui într-un limbaj adecvat.

În numeroase cazuri scopul urmărit prin descrierea algoritmului este legat de necesităţile de comunicare şi documentare. Limbajul folosit în acest caz trebuie să fie apropiat de cel natural, să folosească elemente şi notaţii preluate din algebră. El trebuie să se bazeze pe un sistem de notaţii şi reguli pentru descifrarea univocă şi precisă a algoritmului şi a execuţiei sale. În această situaţie descrierea algoritmului este independentă de limbajele concrete de programare, ceea ce uşurează, într-o anumită măsură, studierea structurii, corectitudinii şi complexităţii sale.

În continuare *un asemenea limbaj va fi denumit limbaj algoritmic sau pseudocod.*

Limbajul algoritmic posedă un vocabular constituit din cuvinte care sînt folosite pentru scrierea comenzilor necesare execuţiei algoritmilor.

O comandă reprezintă o propoziţie cu caracter imperativ, care specifică o atribuire de valoare, efectuarea unor operaţii, calcule, verificări de condiţii, transferuri ale comenzii etc.

În vocabularul limbajului se întîlnesc cuvinte rezervate, ale căror semnificaţii sînt bine precizate, ceea ce uşurează înţelegerea algoritmului. Cuvintele rezervate vor fi scrise cursiv.

Algoritmul descris în acest limbaj trebuie să aibe o denumire. Pentru a specifica acest lucru se va folosi cuvîntul *alg* de la *alg*oritm, urmat de numele algoritmului, de exemplu:

alg cel mai mare divizor comun

În continuare se va folosi cuvîntul *start* pentru a marca începutul secvenţei de instrucţiuni executabile.

Comenzile (instrucţiunile) succesive se pot scrie pe linii succesive sau pe aceeaşi linie, folosind ca delimitator punctul şi virgula (;).

O succesiune de instrucţiuni executabile poartă numele de secvenţă.

Cuvintele *terminat* şi *stop* marchează sfîrşitul unei secvenţe, respectiv — sfîrşitul unui algoritm, conform exemplului următor:

alg nume algoritm
start
secvenţă
terminat
stop

Analiza algoritmilor pune în evidenţă existenţa mai multor structuri: liniare, cu ramificaţii şi cu cicluri.

Algoritmii cu structură liniară sînt reprezentaţi printr-o secvenţă ce constă din comenzi simple înlănţuite liniar sau din secvenţe înlănţuite liniar, conform exemplului de mai jos:

alg nume algoritm
start
secvenţa1
secvenţa2
....
secvenţa N
terminat
stop

Pentru *descrierea ramificaţiilor şi a ciclurilor* se folosesc cuvinte rezervate corespunzătoare.

Ramificaţiile se pot descrie în următoarele moduri:

a) *dacă* condiţie *atunci*
secvenţa1
altfel
secvenţa2
terminat

b) *dacă* condiţie *atunci*

secvenţa1

terminat

Ca exemplu se poate descrie algoritmul determinării acidităţii unei soluţii, folosind hîrtia de turnesol.

alg determinarea acidităţii unei soluţii

start

se toarnă într-o eprubetă 5 ml soluţie;

se introduce în eprubetă hîrtia de turnesol;

dacă hîrtia se înroşeşte *atunci*

soluţia este acidă

altfel

dacă hîrtia devine albastră

atunci

soluţia este o bază

altfel

soluţia este neutră

terminat

terminat

stop

Ciclurile conţin secvenţe de instrucţiuni care se repetă de un număr de ori, în conformitate cu o condiţie impusă. Condiţia poate fi testată la începutul său la sfîrşitul secvenţei, conform exemplelor de mai jos:

a) *cît timp* condiţie *execută*

secvenţă

terminat

b) *ciclează*

secvenţă;

cît timp condiţie

terminat

Se consideră algoritmul de funcţionare a unui regulator de temperatură, bipoziţional, care stabileşte / întrerupe circuitul de alimentare cu energie electrică al unui radiator în funcţie de temperatura mediului şi de valoarea prescrisă de 20°C.

a) *cît timp* temp. mediu<20°C *execută*

stabileşte circuitul;

terminat

b) *ciclează*

stabileşte circuitul;

cît timp temp. mediu< 20°C

terminat

În cazurile în care numărul de cicluri este dinainte cunoscut, se poate folosi un contor. Contorul are o valoare iniţială şi trebuie să ajungă la o valoare finală, prin incrementări cu un pas dat. Valorile iniţială, finală şi pasul sînt cunoscute şi reprezintă numere întregi.

Forma generală a ciclului cu contor este următoarea:

pentru contor=val. iniţială, val. finală, pas *execută*
secvenţă
terminat

Ca exemplu se va prezenta calculul valorii funcţiei $y=5 \cdot x^2$, pentru $5 \leqslant x \leqslant 25$, cu pasul egal cu 1:

alg calculului valorii funcţiei $y=5 \cdot x$
start
pentru i=5, 25, 1 *execută*
$y=5 \cdot x^2$
terminat
stop

Tipuri de date. Marea majoritate a problemelor practice sînt legate de prelucrări ale informaţiei. În cadrul descrierii şi execuţiei unui algoritm trebuie să se evidenţieze informaţia iniţială (datele de intrare), informaţia în curs de prelucrare (rezultatele intermediare) şi informaţia finală (rezultatele). Acestea reprezintă mărimi numerice, grafice etc.

Mărimile sînt de două tipuri: constante, care nu se modifică pe parcursul execuţiei algoritmului şi *variabile,* care iau diverse valori în timpul execuţiei algoritmului.

În descrierea unui algoritm, variabilele vor fi reprezentate prin numele lor, scrise în text cu litere cursive.

Mărimile cu care operează un algoritm pot fi numerice (numere naturale, întregi, reale etc) sau *nenumerice (cuvinte, tabele, texte, grafice, figuri geometrice etc).*

Variabilele numerice vor fi declarate în titlul algoritmului prin precizarea tipului:

întreg i
real x

Variabilele ale căror valori reprezintă cuvinte sau texte vor fi declarate prin *lit*:

lit y

Variabilele numerice pot fi tablouri unidimensionale sau bidimensionale, care reprezintă vectori sau matrici. Elementele acestora vor purta numele vectorului sau matricii precedate de cuvîntul *tab*, de la tablou şi vor avea unul sau doi indici, pentru a specifica poziţia elementelor date.

Astfel, un vector *v*, cu 10 componente reale, va fi declarat ca tablou unidimensional după cum urmează:

real tab v [0 : 9]

O matrice *m*, cu 8 linii şi 8 coloane, constituită din variabile întregi, va fi declarată ca tablou bidimensional ca în exemplul de mai jos, unde indicii iau valori cuprinse între 0 şi 7:

întreg tab m [0: 7, 0: 7]

Precizări privind descrierea algoritmilor în limbajul algoritmic.

Denumirea algoritmului trebuie să conţină declaraţii ale variabilelor cu care se operează. Este necesar a se preciza variabilele care reprezintă argumen-

tele (datele de intrare) şi cele care specifică rezultatele. Pentru acestea se vor folosi cuvintele rezervate: *arg* şi *rez*. Pe parcursul execuţiei algoritmului se mai folosesc şi variabile auxiliare, cu caracter temporar, al căror tip trebuie declarat înainte de utilizare.

În cadrul secvenţelor de instrucţiuni, în mod frecvent se întîlneşte *operaţia de atribuire*, cu următoarea formă generală:

variabila←expresie

***Variabila este dată prin nume.* În urma evaluării expresiei se obţine o valoare, care se atribuie variabilei, al cărei nume constituie termenul stîng al formei de mai sus. Simbolul ← specifică operaţia de atribuire.**

În continuare se vor da unele exemple pentru a ilustra elementele prezentate mai sus.

Exemplu 1. *Algoritmul pentru rezolvarea ecuaţiei de gradul 2:*

$$a \cdot {}^2x + b \cdot x + c = 0$$

```
alg soluţia ec.gr. doi (real a, b, c, x1, x2, lit y)
    arg a, b, c
    rez x1, x2, y
    start
        real D
            D←b²−4ac
            dacă D<0 atunci
                                y←„nu există soluţie"
                       altfel
                       y←„există soluţie"
                                x1←(−b+√D)/2a
                                x2←(−b−√D)/2a
            terminat
    stop
```

Se constată că D reprezintă o variabilă reală de lucru, cu utilizare temporară.

Exemplul 2. *Algoritmul lui Euclid pentru aflarea cmmdc a două numere reale a şi b.*

```
alg cel mai mare divizor comun (real a, b, cmmdc)
    arg a, b
    rez cmmdc
    start
                real x, y
                x←a; y←b cît timp x≠y execută
                                        dacă x>y atunci
                                                        x←x−y
                                                altfel
                                                        y←y−x
                                        terminat
                                terminat
                cmmdc←x
    stop
```

Exemplul 3. *Algoritmul găsirii celui mai mare număr din două numere date* x și y, pe scurt *cmmddn*

```
alg cmmddn (real x, y, z)
    arg x, y
    rez z
    start
        dacă y⩽x atunci
                z←x
            altfel
                z←y
        terminat
    stop
```

Exemplul 4. *Algoritmul găsirii celui mai mare număr dintr-un șir* de n numere date, pe scurt *cmmdnn.*

```
alg cmmdnn (întreg n, real tab x [1 : n], real xmax)
  arg x, n
  rez xmax
  start
      întreg i
        i←1; xmax←0
        cît timp i⩽n execută
              dacă xmax< x[i] atunci
                      xmax←x[i]
              terminat
            i←i+1
  terminat
  stop
```

Exemplul 5. *Algoritmul pentru scrierea tablei înmulțirii.* Tabla înmulțirii va apare ca un tablou cu 9 linii și 9 coloane de forma următoare:

prod[1,1], *prod* [1,2], ..., *prod*[1,9]

....

prod [9,1], *prod* [9,2] .... *prod* [9,9]

unde: *prod* $[i, j] \leftarrow i \times j$.

```
alg tabla înmulțirii (întreg tab prod [1:9, 1:9])
    arg întreg i, j
    rez prod
    start
        i←1
        cît timp i⩽9 execută
                i←1
                cît timp j⩽9 execută
                        prod [i, j]←i×j
                        j←j+1
                terminat
            i←i+1
        terminat
    stop
```

Exemplul 6. *Algoritmul înmulțirii a două matrici A și B, avînd* ca rezultat matricea produs C. Matricea A are $n \times m$ elemente, matricea B are $m \times p$ elemente, iar matricea produs va avea $n \times p$ elemente. Formula de calcul pentru elementele matricei produs este următoarea:

$$C[i, j] = \sum_{j=1}^{m} A[i, j] \times B[j, k], \text{ pentru toți } i=1, n \text{ și } k=1, n.$$

```
alg înmulţire matrici (întreg m, n, p, real tab A[1:n, 1:m], B[1:n,
                       1:p], C[1:n, 1:p])
    arg A, B, m, n, p
    rez C
    start
       întreg i, k, j; real suma
       pentru i=1, n, 1 execută
         pentru k=1, p, 1 execută
                   suma←0
                                 pentru j =1, m, 1 execută
                                         suma←suma+A[i, j]×
                                                  ×B[j, k]
                                 terminat
                   C[i, k]←suma
       terminat
       terminat
stop
```

Încorporarea unor algoritmi existenți în algoritmi de complexitate mai mare reprezintă o practică curent întîlnită în programare, care se materializează prin chemarea de funcții și subrutine. Aceasta asigură o anumită structurare a algoritmului, permițînd o abordare mai simplă a problemelor de mare complexitate.

Exemplu. Se va da un exemplu simplu de folosire a algoritmului cmmddn, pentru a găsi cel mai mare număr din trei numere date (cmmdtn).

```
alg cmmdtn (real a, b, c, d)
  arg a, b, c
  rez d
  start
      real x
      cmmddn (a, b, x)
      cmmddn (x, c, d)
  stop
```

Ca urmare a aplicării algoritmului pe perechea a, b, se obține rezultatul x, reprezentînd cel mai mare număr dintre a și b. În continuare, operația se repetă cu x și c, rezultatul fiind d.

Se constată că în cele două cazuri de aplicare a algoritmului *cmmddn*, argumentele au fost declarate anterior.

Algoritmi pentru manipularea informației grafice. În practică apar probleme ale căror rezultate sînt sub formă de desene, grafice, diagrame etc. Acestea se pot reprezenta pe ecranul dispozitivului de afișare TV sau pe un dispozitiv de înregistrare.

Se presupune că desenul se efectuează pe un ecran, în cadrul unui sistem de coordonate x, y, cu originea (0, 0) în colțul din stînga-jos, al ecranului.

Desenul este realizat prin comenzi date unei „peniţe" imaginare avînd forma unei săgeţi, care se poate deplasa pe ecran cu comenzile următoare: *înainte* (*par* 1), *înapoi* (*par* 2), *dreapta* (*par* 3), *stînga* (*par* 4).

Parametrii *par* 1 şi *par* 2 reprezintă numărul de paşi elementari efectuaţi de peniţă în sensurile date, pe direcţia indicată de vîrful săgeţii.

Parametrii *par* 3 şi *par* 4 specifică numărul de grade cu care se poate roti peniţa, la dreapta sau la stînga, faţă de direcţia curentă.

Lista de comenzi trebuie completată şi cu instrucţiunile: *desenează* şi *nu desena*, pentru a specifica plasarea peniţei pe „hîrtie" şi ridicarea „peniţei" de pe hîrtie.

În starea iniţială peniţa este plasată în punctul de coordonate 0, 0, avînd vîrful în sus.

Exemplu. Desenarea unui triunghi

```
alg triunghi
 start
     desenează
           dreapta (30)
           înainte (5)
           dreapta (120)
           înainte (5)
           stînga (60)
           înapoi (5)
     nu desena
 stop
```

Exemplu. Desenarea unui pătrat cu latura egală cu 5.

```
alg pătrat
 start
     desenează
           înainte (5)
           dreapta (90)
           înainte (5)
           dreapta (90)
           înainte (5)
           dreapta (90)
           înainte (5)
     nu desena
 stop
```

Algoritmul anterior conţine grupuri de instrucţiuni care se repetă, ceea ce sugerează rescrierea lui cu ajutorul instrucţiunii de ciclare.

Exemplu. Desenarea unui pătrat cu latura L.

```
alg pătrat (real L)
 arg L
 start
     întreg i
     i←1 ; desenează
```

cît timp $i \leq 4$ *execută*

înainte (L)

dreapta (90)

$i \leftarrow i+1$

terminat

nu desena

stop

Trecerea în revistă a limbajului algoritmic, permite însuşirea primelor noţiuni de programare în limbaje de nivel înalt. Fiecare limbaj de programare are particularităţile sale, ceea ce îi poate conferi o serie de avantaje, pentru anumite tipuri de aplicaţii. În cadrul limbajului algoritmic nu s-au tratat problemele de citire / scriere a informaţiilor de la / la un terminal. De asemenea, structurile de date examinate, reprezentînd scalari (întregi, reali), tablouri uni şi bidimensionale (vectori şi matrici de numere întregi sau reale) şi texte constituie structurile cele mai simple întîlnite în practică.

După cum s-a mai menţionat, calculatorul HC-85 dispune de un interpretor memorat pentru limbajul BASIC şi de o serie de interpretoare şi compilatoare, pentru limbajele LOGO, Pascal, C, Microprolog, LISP, Fortran.

În lucrarea de faţă vor fi prezentate limbajele BASIC şi LOGO.

Partea a IV-a

PROGRAMAREA ÎN LIMBAJUL BASIC PE CALCULATORUL HC-85

Capitolul 8 | Caracteristicile şi elementele limbajului BASIC

8.1. Noţiuni introductive

La denumirea de BASIC s-a ajuns prin păstrarea iniţialelor denumirii complete din limba engleză „Beginners All purpose Symbolic Instruction Code" (Codul instrucţiunilor simbolice pentru începători, utilizabil în orice scop).

După cum îi spune şi numele, el este destinat cu precădere începătorilor şi are diverse şi multiple utilizări.

Este uşor, simplu, asimilabil într-un timp scurt, fără a necesita o pregătire prealabilă specială.

BASIC este unul dintre cele mai răspîndite limbaje de programare din lume.

Limbajul de programare este conceput special în vederea realizării înţelegerii dintre om şi maşină. El poate fi sugestiv asimilat cu „limba" pe care o înţelege un calculator. Numai cunoscînd un astfel de limbaj de programare se poate beneficia de toate facilităţile oferite de un calculator personal.

Limbajul de programare BASIC este alcătuit dintr-un set de instrucţiuni şi comenzi, alături de toate regulile de formare şi folosire a acestora.

O instrucţiune sau o comandă se recunoaşte prin „cuvîntul cheie" ce reprezintă denumirea sa. Această denumire este aleasă întotdeauna sugestiv şi provine din prescurtarea cuvîntului asociat din limba engleză.

O comandă şi o instrucţiune pot avea aceeaşi denumire, dar deosebirile dintre ele sînt fundamentale şi constau în modul de memorare, de execuţie şi de păstrare în memorie.

Imediat după introducere, o *instrucţiune* se memorează, după care se execută imediat sau după un anumit timp, o dată sau de cîte ori se dovedeşte a fi necesară; ea se păstrează în memorie pînă la eventuala ei ştergere ce survine numai la cerere.

În cazul unei *comenzi*, după introducere urmează imediat execuţia sa şi aceasta numai o singură dată, după care execuţia nu mai poate fi reluată decît printr-o nouă introducere deoarece ea nu se păstrează în memorie.

Instrucţiunile şi comenzile se supun unor reguli de sintaxă (de scriere) şi unor reguli de logică (de execuţie).

Mulţimea ordonată şi finită a instrucţiunilor / comenzilor, ce soluţionează o problemă, formează un *program*.

Programul se bazează pe un algoritm.

Algoritmul constă în totalitatea raţionamentelor matematice şi logice, făcute pas cu pas, în scopul rezolvării unei probleme cu ajutorul calculatorului.

Algoritmul se transpune grafic printr-o schemă logică.

Schema logică este mulţimea tuturor simbolurilor standard cu o semnificaţie predefinită prin care se reprezintă paşii parcurşi în vederea transpunerii unei probleme într-un limbaj de programare.

Schema logică este cu atît mai necesară cu cît problema propusă spre rezolvare este mai complexă.

Ansamblul programelor destinate rezolvării unor clase de probleme este cunoscut sub numele de *soft (software) de aplicaţie*.

În afara acestei categorii, există programe fără de care utilizarea calculatorului nu este posibilă.

Ansamblul acestor tipuri de programe este cunoscut sub numele de *soft de bază*.

BASIC face parte din componenta soft de bază. El este livrat odată cu calculatorul personal şi se încarcă automat în memorie prin simpla pornire a sistemului.

Elementele precizate sub denumirea de soft de aplicaţie şi soft de bază se numesc pe scurt, soft. Acesta este un produs intelectual ce nu poate exista în afara echipamentului fizic pentru care este conceput.

Echipamentul fizic sau calculatorul împreună cu toate perifericele (televizor, casetofon, etc.) este cunoscut sub numele de *hard (hardware)*.

Componentele hard şi soft alături de factorul uman alcătuiesc un sistem de prelucrare automată a datelor (sistem informatic).

Cele trei componente ale unui sistem de prelucrare automată a datelor sînt într-o continuă interacţiune şi dinamică.

Ele alcătuiesc o unitate, ce înmagazinează multă inteligenţă, experienţă şi muncă.

Calculatorul este una din uneltele reprezentative ale secolului XX, ce merită să fie explorată cu atenţie, perseverenţă şi devotament, pentru a ajunge un instrument popular, de folos în orice domeniu de activitate.

Este clar că, fără cunoaşterea unui limbaj de programare, calculatorul, deşi oferă multe posibilităţi, devine un obiect inutilizabil, fără nici o eficienţă. În sprijinul depăşirii acestui obstacol, capitolul prezent îşi propune iniţierea în limbajul de programare BASIC.

Sînt necesare cîteva precizări.

Pentru că în lume s-au produs diverse tipuri de calculatoare personale, circulă şi multe variante ale limbajului BASIC.

La o variantă a limbajului BASIC se ajunge prin preluarea unui subset de instrucţiuni din limbajul BASIC standard. Selectarea şi implementarea diferită a instrucţiunilor / comenzilor unui limbaj de programare este generată de multitudinea diferenţelor existente între parametrii fizici ai echipamentelor.

Limbajul BASIC prezentat aici este implementat pe calculatorul personal românesc HC-85, compatibil cu calculatoarele din gama SPECTRUM —SINCLAIR.

8.2. Tastatura calculatorului HC-85

Alfabetul utilizat de HC-85 cuprinde 256 simboluri.

Simbolurile pot fi:

— simboluri simple (litere, cifre, simboluri speciale etc.);

— simboluri compuse (cuvinte cheie ale instrucţiunilor sau comenzilor, nume de funcţii etc.).

Fiecare simbol se găseşte imprimat pe o tastă.

Tastatura calculatorului HC-85 cuprinde 40 de taste şi este similară cu cea a unei maşini de scris: literele şi cifrele sînt în aceleaşi poziţii cu excepţia literelor Q, Z şi M. Tastele sînt plasate pe 4 rînduri, cîte 10 pe linie.

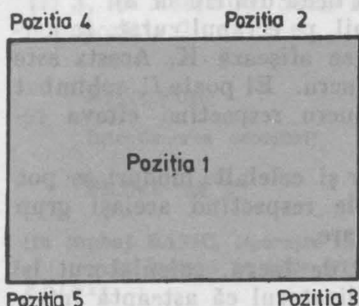


Fig. 8.1.

În centrul tastei apare scris un simbol, în stînga şi dreapta sus sau jos alte simboluri, conform poziţiilor din figura 8.1.

Trebuie reţinut faptul că, pentru a obţine toate simbolurile alfabetului BASIC, unele taste au pînă la şase semnificaţii.

Aceste semnificaţii, potrivit celor scrise în poziţiile 1—5 din figură, sînt selectate conform modului de lucru în care se găseşte calculatorul, prin apăsarea tastei respective simultan cu tasta:

— CAPS SHIFT (cu prescurtarea CS — prima tastă din rîndul 4);

— SYMBOL SHIFT (cu prescurtarea SS — penultima tastă din rîndul 4).

Dacă o tastă este apăsată mai mult de 2—3 secunde, simbolul începe să se repete.

Semnificaţia tastelor apăsate apare pe rînd în partea de jos a ecranului, fiecare caracter fiind inserat pe locul cursorului.

Cursorul este săgeata neastîmpărată de pe ecran, care se deplasează singură pe măsură ce alte date se introduc şi indică locul unde se face viitoarea tipărire sau înscriere. Cursorul poate fi mutat la stînga cu ← (CAPS SHIFT şi 5) sau la dreapta cu → (CAPS SHIFT şi 8). Caracterul din stînga cursorului poate fi şters cu DELETE (CAPS SHIFT şi 0).

La înscrierea simbolurilor pe tastatură au fost folosite următoarele prescurtări:

RAND	—	în loc de	RANDOMIZE
BRGT	—	în loc de	BRIGHT
INV	—	în loc de	INVERSE
CR	—	în loc de	ENTER
CS	—	în loc de	CAPS SHIFT
SS	—	în loc de	SYMBOL SHIFT
SCR	—	în loc de	SCREEN
CONT	—	în loc de	CONTINUE

8.3. Modurile de lucru

Modul de lucru este modul prin care acţionînd o tastă, în memorie este introdus unul din simbolurile scrise pe ea în poziţia 1—5.

Calculatorul HC-85 are 5 moduri de lucru, desemnate prin literele K, L, C, E şi G.

Modul de lucru este iniţial afişat în partea stîngă jos a ecranului. Litera ce îl desemnează se deplasează automat pe ecran pe măsură ce se introduc date.

În practică, după conectarea la reţea, calculatorul se prezintă (fig. 8.2).

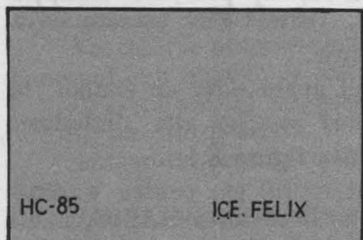


Fig. 8.2.

După această manevră trebuie acţionată tasta CR (ultima tastă din rîndul 3).

Imperturbabil, pe ecranul curat, în partea stîngă jos, se afişează K. Acesta este primul mod de lucru. El poate fi schimbat în alt mod de lucru respectînd cîteva reguli de tastare.

La rîndul lor şi celelalte moduri se pot schimba între ele respectînd acelaşi grup de reguli de tastare.

Prin modul de lucru, calculatorul îşi atenţionează utilizatorul că aşteaptă introducerea unui anumit tip de date. Mai mult, dacă nu se introduc datele aşteptate (din punct de vedere al logicii limbajului utilizat), pe ecran, acolo unde este greşit, apar semne de întrebare şi nu se poate trece mai departe decît dacă se fac corecţiile cuvenite.

8.3.1. Modul de lucru K

Modul K (keyword=cuvînt cheie) apare atunci cînd se afişează o comandă (ex: LIST, RUN, SAVE, PRINT etc.) sau o instrucţiune (ex. 100 PRINT, 20 INPUT, 66 LET etc). Aceasta se întîmplă la începutul unei linii program, după simbolul „ : “ (ce separă instrucţiunile de pe aceeaşi linie) sau după un THEN (cuvînt cheie al unei instrucţiuni de comparaţie).

În modul K, tastele literale sînt interpretate ca nişte cuvinte cheie, potrivit notaţiilor din poziţia 3, iar tastele numerice sînt interpretate ca simple numere, potrivit notaţiilor din poziţia 1.

8.3.2. Modul de lucru L

Modul L (letters = litere) alternează cu modul K; imediat ce a fost introdus un cuvînt cheie, modul de lucru K se schimbă automat în L.

În modul L, simbolul scris în poziţia 1 pe orice tastă apare prin simpla acţionare a acesteia. În cazul unei taste literale apare litera mică a alfabetului.

EXEMPLU:

Introducerea comenzii

PRINT 25

(în limbaj BASIC, tipăreşte numărul 25) se face astfel:

— la început de linie este afişat modul de lucru K, se acţionează tasta P ceea ce înseamnă introducerea simbolului din poziţia 3, adică PRINT;

— modul de lucru se schimbă automat în L, iar acţionarea tastelor numerice 2 şi 5 conduce la introducerea simbolurilor scrise în poziţia 1 a tastelor numerice specificate.

Atît în modul L cît şi în modul K, acţionarea simultană a lui SYMBOL SHIFT (SS) şi a unei taste numerice este interpretată drept simbolul din poziţia 3, iar SYMBOL SHIFT (SS) simultan cu o tastă literală produce simbolul din poziţia 2.

EXEMPLU:

Introducerea comenzii

PRINT "25 $"

(în limbaj BASIC, tipăreşte textul 25 $) se face astfel:

— la început de linie este afişat modul de lucru K, se acţionează tasta P şi rezultă PRINT;

— modul de lucru devine automat L, prin acţionarea simultană a tastelor SS şi P se produce scrierea simbolului " (ghilimele), din poziţia 2 a tastei literale P;

— în continuare modul de lucru rămîne tot L, se acţionează pe rînd tastele numerice 2 şi 5 pentru afişarea numărului 25;

— fiind în modul L, pentru afişarea semnului $, se acţionează simultan tasta numerică 4 şi tasta SS;

— în final se apasă încă o dată simultan pe tastele SS şi P pentru închiderea ghilimelelor.

Acţionarea simultană a lui CAPS SHIFT şi a unei taste numerice, în modul de lucru L, este interpretată ca simbolul specificat în poziţia 4, ca de exemplu EDIT, CAPS LOCK, DELETE ş.a.m.d.

EXEMPLU:

Introducerea comenzii

DELETE

(în limbaj BASIC, şterge un număr de caractere) se face astfel:

— pentru ştergerea unui caracter se acţionează simultan, o singură dată, tasta 0 (zero) şi CS;

— pentru ştergerea unui grup de caractere se acţionează simultan tastele 0 (zero) şi CS de un număr de ori egal cu numărul caracterelor componente ale acelui grup.

Acţionarea unei taste literale simultan cu CAPS SHIFT, în modul de lucru K, nu are nici un efect (sau tasta este interpretată conform poziţiei 3), iar în modul de lucru L produce conversia literelor mici în litere mari.

EXEMPLU:

Introducerea comenzii

PRINT A+B

(în limbaj BASIC, tipărește suma numerelor existente în A și B) se face astfel:

— fiind în modul K se tastează P și pe ecran apare PRINT;

— modul de lucru devine automat L, se acționează simultan tastele CS și A din care rezultă litera mare A;

— tastarea simultană a lui SS și K dă semnul +;

— următoarea tastare se compune din apăsarea simultană a tastelor CS și B din care rezultă litera mare B.

8.3.3. Modul de lucru C

Modul C (capitals=majuscule) este o variantă a modului de lucru L, în care scrierea se face cu litere mari.

Tasta CAPS LOCK (acționarea simultană a tastelor CS și 2) determină trecerea din modul L în modul C. Numai dacă este acționată din nou tasta CAPS LOCK se revine din modul C în modul L.

EXEMPLU:

Introducerea comenzii

PRINT A+B

se poate face și astfel:

— primul pas este identic cu cel din exemplul anterior;

— modul de lucru devine automat L, iar prin tastarea simultană a lui CS și 2 se schimbă în C, după care se apasă direct tasta A pentru tipărirea literei mari A;

— pentru semnul + se procedează ca în exemplul anterior;

— modul de lucru C se păstrează în continuare, iar prin apăsarea directă a tastei B, se obține ultimul caracter dorit, litera mare B.

8.3.4. Modul de lucru E

Modul de lucru E (extended — extins) este utilizat pentru a obține simboluri noi, în special alte instrucțiuni sau comenzi. Pentru a intra în acest mod se acționează simultan ambele SHIFT-uri (CS și SS), iar anularea acestui mod se face automat după prima tastare.

În acest mod, apăsarea unei taste literale generează simbolul scris în poziția 4, iar dacă tasta este apăsată împreună cu o tastă SHIFT se generează simbolul scris în poziția 5 a tastei.

EXEMPLU:

Introducerea comenzii

READ nr

(în limbaj BASIC, citește variabila cu numele „nr") se face astfel:

— în stînga jos pe ecran apare tipărit modul K, el se schimbă în modul E prin apăsarea simultană a tastelor CS și SS;

— în modul de lucru E, apăsarea tastei A conduce la obţinerea simbolului din poziţia 4 a tastei, respectiv READ;
— modul de lucru se schimbă automat în L, apăsarea pe rînd a tastelor N şi R conduce la obţinerea caracterelor dorite, respectiv variabila cu numele „nr".

EXEMPLU:

Introducerea comenzii

BEEP 1,3

(în limbaj BASIC, sunetul corespunzător unei note muzicale) se face astfel:
— se transformă modul K în modul E prin apăsarea simultană a tastelor CS şi SS;
— în modul de lucru E, apăsarea simultană a tastelor Z şi CS (sau SS) conduce la obţinerea simbolului din poziţia 5 a tastei Z, respectiv BEEP;
— modul de lucru se schimbă automat în L, iar pentru obţinerea numărului 1 se apasă pe tasta 1;
— modul de lucru nu se schimbă, este tot L, se apasă simultan pe tasta N şi SS obţinîndu-se simbolul virgulei;
— ultima tastare, în modul de lucru L, constă în apăsarea tastei 3 pentru introducerea numărului 3.

Apăsarea unei taste numerice în modul de lucru E generează o comandă dacă este acţionată împreună cu SYMBOL SHIFT şi o secvenţă de control a culorii dacă este acţionată singură.

8.3.5. Modul de lucru G

Modul G (graphics — grafice) se obţine prin acţionarea tastelor CAPS SHIFT şi 9. Anularea acestui mod de lucru se face acţionînd din nou tastele CAPS SHIFT şi 9 sau numai tasta 9.

O tastă numerică dă un mozaic grafic predefinit (în afara tastelor 0 şi 9) şi orice tastă literală în afara de V, W, X, Y şi Z generează un simbol grafic definit de utilizator.

Acţionarea lui V, W, X, Y sau Z în modul de lucru G conduce la introducerea unui simbol deja stabilit care reprezintă cea de a şasea semnificaţie a acestor taste:

V — RND
W — INKEY $
X — PI
Y — FN
Z — POINT

EXEMPLU:
Introducerea comenzii

DRAW 10, 50, PI

(în limbaj BASIC, trasează conturul unui semicerc) se face astfel:
— la început de linie este afişat modul de lucru K, se acţionează tasta W şi se obţine DRAW;
— modul de lucru devine automat L, acţionarea pe rînd a tastelor 1 şi 0 introduce numărul 10;

— în continuare modul de lucru este tot L, se acţionează simultan tasta SS şi N pentru obţinerea simbolului virgulă;

— modul de lucru fiind tot L se procedează ca înainte pentru tipărirea simbolurilor 5 şi 0;

— prin acţionarea simultană a tastelor CS şi 9 modul de lucru devine G, iar simpla apăsare a tastei X tipăreşte simbolul PI.

În scopul fixării tuturor posibilităţilor de transformare a modurilor de lucru între ele în vederea obţinerii tuturor simbolurilor specificate pe tastatura calculatorului HC-85, a rezultat necesitatea sistematizării celor prezentate pînă în acest moment. Pentru aceasta s-au realizat fig. 8.3. şi tabelul 8.1.

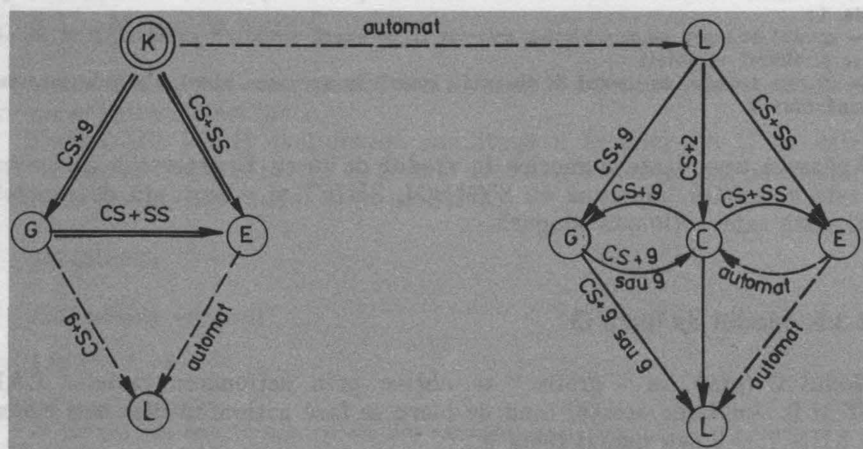


Fig. 8.3.

În figura 8.3 sînt indicate cu ajutorul săgeţilor toate direcţiile posibile de schimbare a unui mod de lucru în alt mod de lucru, potrivit acţionării corespunzătoare a tastei/tastelor specificate chiar de-a lungul săgeţilor, cu precizările:

a) modul de lucru K este dublu încercuit pentru a scoate în evidenţă că acesta este întotdeauna primul mod de lucru pe care îl afişează calculatorul;

b) imediat după tastarea unui simbol, modul de lucru K se transformă automat în L, existînd însă posibilitatea schimbării lui înainte de aceasta, conform săgeţilor duble;

c) numai după tastarea cel puţin a unui simbol într-un anumit mod de lucru, se poate trece în alt mod de lucru, conform săgeţilor întrerupte.

d) fie că s-a tastat sau nu un simbol într-un anumit mod de lucru, se poate trece în alt mod de lucru, conform săgeţilor pline (continue);

e) fiind în modul de lucru C, se poate trece în modul G sau E, dar în momentul schimbării acestora se revine tot în modul C.

Tabelul 8.1 conţine corespondenţa dintre simbolul precizat pe tastă într-o anumită poziţie şi un mod de lucru, epuizînd pe rînd atît cele 5 moduri de lucru existente cît şi toate cele 5 sau 6 semnificaţii ale unei taste.

Tabelul 8. 1.

MOD K	tastă numerică	poz. 1
	tastă literală	poz. 3
MOD L	tastă numerică	poz. 1
	tastă numerică + CS	poz. 4
	tastă numerică + SS	poz. 3
	tastă literală	poz. 1 (literă mică)
	tastă literală + CS	poz. 1 (literă mare)
	tastă literală + SS	poz. 2
MOD C	tastă numerică	poz. 1
	tastă literală	poz. 1 (literă mare)
MOD E	tastă numerica (1-7 și 0)	secv. de control a culorii
	tastă numerică + SS	poz. 5
	tastă literală	poz. 4
	tastă literală CS sau SS	poz. 5
MOD G	tastă numerică (1-8)	poz. 2
	tastă literală (cu excepția lui V,W,X,Y,Z)	poz. 1 (literă mare)

8.4. Alfabetul limbajului BASIC

Prin acționarea tuturor tastelor calculatorului HC-85 și a combinațiilor posibile potrivit celor 5 moduri de lucru, se obțin toate cele 256 simboluri componente ale alfabetului BASIC.

Alfabetul limbajului BASIC se compune din simboluri simple și simboluri compuse.

Un simbol simplu este format dintr-un singur caracter obținut în urma unei singure tastări.

Un simbol compus este format dintr-un grup de caractere obţinut (atenţie!) în urma unei singure tastări.

Diferenţa dintre cele 2 tipuri de simboluri nu constă numai în numărul de caractere ci şi în faptul că un simbol compus nu este echivalent cu simbolul obţinut prin tastarea caracter cu caracter.

Din categoria simbolurilor simple fac parte literele, cifrele, semnele de punctuaţie, semnele operaţiilor matematice etc.

Din categoria simbolurilor compuse fac parte cuvintele cheie ale instrucţiunilor sau comenzilor, funcţiile matematice etc.

Cu simbolurile simple se construiesc constante, variabile sau expresii, elemente atît de necesare definirii argumentelor instrucţiunilor şi comenzilor BASIC. Cuvintele cheie alături de argumente potrivite generează setul de instrucţiuni şi comenzi capabile să realizeze dialogul dintre utilizator şi calculator.

Înainte de expunerea efectivă a instrucţiunilor şi comenzilor este absolut necesară familiarizarea cu elementele ce au sens în limbajul BASIC.

8.4.1. Setul de caractere

Setul de caractere este format din totalitatea simbolurilor ce pot face parte dintr-o instrucţiune sau comandă în limbajul de programare BASIC.

Setul de caractere este format din:

a) — litere;
b) — cifre;
c) — operatori aritmetici;
d) — operatori de relaţie;
e) — operatori logici;
f) — semne de punctuaţie;
g) — semne speciale.

a) Litere

Literele sînt în număr de 26 şi sînt literele cunoscute, de la a la z, mici şi mari.

b) Cifre

Cifrele sînt în număr de 10 şi sînt cifrele sistemului zecimal, de la 0 la 9.

c) Operatori aritmetici

Operaţiile aritmetice executate de calculator au următorii operatori:

+ pentru	— adunare şi semnul algebric + ;
—	— scădere şi semnul algebric — ;
*	— înmulţire;
/	— împărţire;
↑	— ridicare la putere.

Ordinea de execuţie a operaţiilor aritmetice este cea cunoscută din matematică:

prioritatea 1	ridicarea la putere;
prioritatea 2	înmulţirea şi împărţirea;
prioritatea 3	adunarea şi scăderea.

Dacă există numai operații cu aceeaşi prioritate ordinea de efectuare este de la stînga la dreapta, inclusiv pentru ridicarea la putere.

EXEMPLE:

2 ↑ 3 ↑ 2 = 8 ↑ 2 = 64
3 ↑ 2 ↑ 3 = 9 ↑ 3 = 729

Ordinea de execuţie poate fi schimbată cu ajutorul parantezelor.

d) Operatori de relaţie

Relaţiile de ordine executate de calculator au următorii operatori de relaţie:
= pentru relaţia de — egalitate;
< — mai mic;
<= — mai mic sau egal;
> — mai mare;
>= — mai mare sau egal;
< > — diferit de.

Relaţiile de ordine într-o mulţime numerică sînt relaţiile de egalitate sau inegalitate cunoscute din matematică.

În mulţimea şirurilor de caractere, ca relaţie de ordine este folosită ordinea alfabetică cunoscută.

e) Operatori logici

Operaţiile logice executate de calculator au următorii operatori:
AND pentru — intersecţie;
OR — reuniune;
NOT — negaţie.

Operaţiile logice simbolizează operaţiile din logica matematică obişnuită.
Ordinea de execuţie a operaţiilor logice este cea cunoscută din matematică:
NOT
AND
OR

În expresii complicate ce conţin toate tipurile de operatori ordinea de execuţie este următoarea:
prioritatea 1 — operatori logici;
prioritatea 2 — operatori de relaţie;
prioritatea 3 — operatori aritmetici.

Dacă se doreşte să se schimbe această ordine se folosesc parantezele.
Între operatori cu acelaşi nivel de prioritate, ordinea de execuţie este de la stînga la dreapta.

f) Semne de punctuaţie

Dintre semnele de punctuaţie fac parte simbolurile cunoscute pentru:

. — punct
: — două puncte
; — punct şi virgulă
, — virgulă
' — apostrof
" — ghilimele
! — semnul exclamării
? — semnul întrebării

g) Semne speciale

Dintre semnele speciale fac parte simbolurile cunoscute pentru:

	—	spaţiu liber, blanc (b̸)
()	—	paranteze rotunde
[]	—	paranteze drepte
CR	—	retur de car
%	—	procent
$	—	dolar
#	—	diez

Numărul semnelor speciale este mai mare decît cel specificat. Afirmaţia se demonstrează, de exemplu, prin acţionarea tastelor numerice de la 1 la 8, în modul de lucru G.

8.4.2. Constante

Constantele sînt valori fixe. Ele sînt de tip numeric sau alfanumeric.

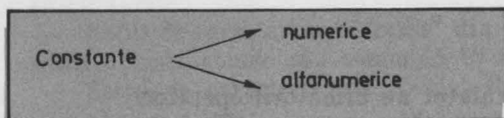


Fig. 8.4.

1°. Constante numerice

Constantele numerice sînt şiruri de cifre negative sau pozitive.

În limbajul BASIC constantele numerice se pot exprima în sistemele de numeraţie:

— zecimal (numărul este reprezentat cu ajutorul cifrelor de la 0 la 9);

— binar (numărul este reprezentat cu ajutorul cifrelor 0 şi 1);

— hexazecimal (numărul este reprezentat cu ajutorul cifrelor de la 0 la 9 şi a literelor A, B, C, D, E, F cu semnificaţia A=10, B=11, C=12, D=13, E=14, F=15).

În cazul particular în care constanta este scrisă în sistemul hexazecimal, s-a stabilit convenţia de a fi completată în partea dreaptă cu simbolul &.

Constantele numerice la rîndul lor pot fi, potrivit numerelor pe care le conţin, de două tipuri:

— constante de tip întreg;

— constante de tip real.

Constantele de tip întreg sînt valori întregi.

EXEMPLE: —10
0
1987

Constantele de tip real sînt valori reale în care virgula se reprezintă prin punct.

EXEMPLE:
19.88
0.1988
—0.1988

Reprezentarea constantelor în exemplele anterioare a fost făcută în scriere obişnuită. Ea mai poartă şi numele de scriere în virgulă fixă.

Unele probleme practice şi ştiinţifice impun folosirea unor numere foarte mari sau foarte mici. De exemplu:

a) distanţa de la Pămînt la Lună este de 384.000.000 m;
b) distanţa de la Pămînt la Soare este de 150.000.000.000 m;
c) un gram de hidrogen conţine un număr foarte mare de atomi de hidrogen şi anume:
   6023 00000000000000000000, deci 6023 urmat de 20 de zerouri;
d) un gram de apă conţine
   33 000000000000000000000 molecule de apă, deci 33 urmat de 21 de zerouri;
e) masa unei molecule de apă este de
   3 : 100000000000000000000000000 kg, deci 3 supra 1 urmat de 26 de zerouri.

După cum se vede este foarte incomod de a scrie şi de a lucra cu aceste numere. Pe de altă parte numerele scrise în acest mod ocupă un loc foarte mare în memoria calculatorului. Considerentele specificate au condus la necesitatea găsirii unui alt mod de scriere pentru utilizarea mai comodă a acestor numere.

Acest mod de scriere foloseşte puterile lui 10, astfel că numerele se pot scrie astfel:

a) $384 * 10^{6}$m;
b) $15 * 10^{10}$ sau $1,5 * 10^{11}$ m;
c) $6023 * 10^{20}$ sau $6,023 * 10^{28}$ atomi, (numărul scris sub ultima formă şi notat cu N se utilizează în chimie sub denumirea de numărul lui Avogadro);
d) $33 * 10^{21}$ sau $0,33 * 10^{23}$ molecule de apă;
e) $3 * 10^{-26}$ kg, notînd 1/10 cu 10^{-1}.

Această scriere comodă pentru manipulare, dar şi economică din punct de vedere al reprezentării numerelor în memorie, se numeşte scriere cu exponent (pe scurt scriere cu e).

Orice număr se poate reprezenta ca produsul dintre un număr şi puteri ale lui 10.

Pentru un număr există mai multe variante de a fi reprodus cu ajutorul puterilor lui 10 şi de aceea scrierea cu e se numeşte şi scriere în virgulă mobilă.

În reprezentarea cu e a numerelor, 10 nu apare explicit.

Numerele din exemplele date se reprezintă cu ajutorul scrierii cu e astfel:

a) 384e6 sau 38.4e7 sau 3.84e8 sau 0.384e9;
b) 15e10 sau 1.5e11 sau 0.15e12;
c) 6023e20 sau 6.023e23 sau .....;
d) 33e21 sau 0.33e23;
e) 3e-26 sau 0.3e-27.

Constante numerice → în scriere obişnuită sau scriere în virgulă fixă
Constante numerice → în scriere cu e sau scriere în virgulă mobilă

Fig. 8.5.

Cel mai mare număr ce poate fi reprezentat în memoria calculatorului este numărul:

$$4294967295 = 2^{32} - 1$$

2° Constante alfanumerice

Constantele de tip alfanumeric sînt şiruri de caractere. Prin şir de caractere se înţelege un şir format din orice caracter prezent pe tastatura calculatorului încadrat între ghilimele.

Dacă se doreşte să se tipărească în text caracterul ghilimele atunci el trebuie dublat.

EXEMPLE de constante alfanumerice:

"abc"
"calculator"
"linie + coloana"
"CEL MAI MARE NUMĂR"
"A 1 +A 2"
"234"

8.4.3. Variabile

Variabila este un nume (o etichetă) care se dă unui număr sau unui şir de caractere. Numele este ales de utilizator. El este format dintr-o succesiune de litere şi / sau cifre căruia i se atribuie în diferite momente ale execuţiei unui program, diferite valori.

Variabilele sînt ca şi constantele de două tipuri: numerice sau alfanumerice.

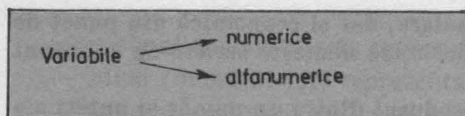


Fig. 8.6.

1° Variabile numerice

Versiunea BASIC a calculatorului HC-85 admite pentru variabilele numerice nume formate din oricîte caractere (litere sau cifre), care încep cu o literă. Printre caractere poate fi şi blancul, care este însă ignorat. Prezenţa lui face variabila mai uşor de citit. Sistemul face filtrarea literelor mari, astfel încît, atît litera mare cît şi litera mică corespunzătoare sînt interpretate la fel.

Este recomandat ca numele unei variabile să fie sugestiv fără însă a crea confuzii şi cît se poate de scurt pentru a fi uşor de manipulat.

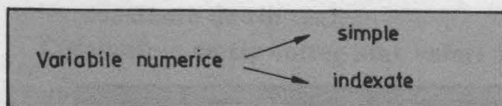


Fig. 8.7.

Variabilele numerice sînt simple sau indexate.

Variabila numerică simplă este un nume căruia i se atribuie la un moment dat o valoare număr real.

EXEMPLE de variabile numerice:

a) R — în accepţiunea unui program ea poate reprezenta raza unui cerc;
— poate lua pe rînd valorile:
0
10.3
28.5702
88

b) LAT — în accepțiunea unui program ea poate reprezenta latura unui pătrat;

— poate lua pe rînd valorile:

0.5
1
7.88
65

Variabila numerică indexată este un nume căruia i se atribuie la un moment dat un şir finit de valori numerice. Numărul valorilor din şir este dat de indicii variabilei.

Un indice este la rîndul său o variabilă numerică. El poate lua valori numere naturale, de la 1, 2, 3. ... pînă la 100 sau mai mare, dar în practică nu este nevoie de un număr aşa de mare de indici.

EXEMPLE de variabile numerice indexate:

a) D(I) **I=1,2** **— în accepțiunea unui program conține cele două dimensiuni ale unui dreptunghi;**
— poate lua pe rînd valorile:
D(1) =15 (lungimea)
D(2) =3 (lățimea)
sau
D(1) =7.5
D(2) =2.3

b) s(k) **k=1,...,5** **— în accepțiunea unui program conține un şir de 5 numere naturale;**
— poate lua pe rînd valorile:
s (1) =1 (primul element al şirului)
s (2) =5 (al doilea element al şirului)
s (3) =0 (al treilea element al şirului)
s (4) =12 (al patrulea element al şirului)
s (5) =2 (al cincilea element al şirului)
sau
s (1) =18
s (2) =1
s (3) =1
s (4) =1
s (5) =0

c) M (I, J) **I=1, 2** **J=1, 2, 3** **— în accepțiunea unui program este un tablou cu 2 linii şi 3 coloane ce conține numere între 0 şi 1;**
— poate lua pe rînd valorile:

	coloana 1	coloana 2	coloana 3
linia 1	M(1,1) =0.1	M(1,2) =0.01	M(1,3) =0.0001
linia 2	M(2,1) =1	M(2,2) =0.7	M(2,3) =0.4

sau

linia 1	M(1,1) =1	M(1,2) =1	M(1,3) =1
linia 2	M(2,1) =0	M(2,2) =1	M(2,3) =0

Este util de reținut că o variabilă numerică simplă poate avea acelaşi nume cu o variabilă numerică indexată fără a se genera confuzii în bunul mers al unui program.

2° Variabile alfanumerice

Versiunea BASIC a calculatorului HC-85 admite pentru variabilele alfanumerice nume formate din oricîte caractere (litere sau cifre), ce încep cu o literă şi se termină cu caracterul $.

Variabilele alfanumerice sînt simple sau indexate.

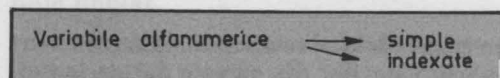


Fig. 8.8.

Variabila alfanumerică simplă este un nume căreia i se atribuie la un moment dat o valoare şir de caractere.

EXEMPLE de variabile alfanumerice:

a) a$ — **poate lua pe rînd valorile:**
ABC
C2M
???

b) TRA-1 $ — **poate lua pe rînd valorile:**
5
DA
—.2

Variabila alfanumerică indexată este un nume format dintr-un singur caracter căruia i se atribuie la un moment dat un şir finit de valori şir de caractere.

Indicii trebuie să îndeplinească aceleaşi condiţii ca cei ai unei variabile numerice indexate.

EXEMPLE de variabile alfanumerice indexate:

a) T$(P) — **în accepţiunea unui program este un şir de caractere;**
P=1,2,3 — **poate lua pe rînd valorile:**
T$ (1) ="ABC" (primul şir al şirului)
T$ (2) ="DEF" (al doilea şir al şirului)
T$ (3) ="GHI" (al treilea şir al şirului)
sau
T$ (1) ="1—A"
T$ (2) ="2—B"
T$ (3) ="3—B"

b) m (u, v) — **în accepţiunea unui program este un tablou cu 2 linii şi 3 coloane de**
u =1,2,3,4 **şiruri de caractere;**
v =1,2 — **poate lua pe rînd valorile:**

	coloana 1	coloana 2
linia 1	m$ (1,1) ="IONESCU DAN"	m$ (1,2) ="BUCURESTI"
linia 2	m$ (2,1) ="IONESCU ION"	m$ (2,2) ="CONSTANŢA"
linia 3	m$ (3,1) ="IONESCU MIRCEA"	m$ (3,2) ="IASI"
linia 4	m$ (4,1) ="IONESCU VLAD"	m$ (4,2) ="BRASOV"
	sau	
linia 1	m$ (1,1) ="ADAM STELA"	m$ (1,2) ="LIC. 1"
linia 2	m$(2,1) ="BOBESCU RADU"	m$ (2,2) ="LIC. 21"
linia 3	m$ (3,1) ="COSTIN MARCEL"	m$ (3,2) ="LIC. 18"
linia 4	m$ (4,1) ="DORIN ANA"	m$ (4,2) ="LIC. 5"

8.4.4. Expresii

Expresia este un grup de simboluri unite printr-o operaţie ce poate fi evaluat. Expresiile pot fi numerice sau alfanumerice, după tipul simbolurilor din grup.

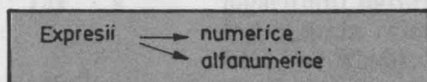


Fig. 8.9.

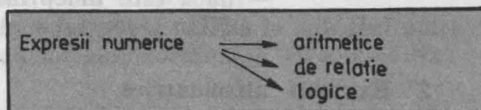


Fig. 8.10.

1° Expresii numerice

Expresiile numerice sînt compuse din constante numerice, variabile simple, variabile indexate sau combinaţii ale acestora separate prin operatori aritmetici, operatori de relaţie sau operatori logici.

Expresia aritmetică este expresia numerică care conţine operatorii aritmetici cunoscuţi : +, —, *, /. ↑

EXEMPLE de expresii aritmetice:

A+B
2 ↑ C
5 ((A+3/C)—B * D)

Expresia de relaţie este expresia numerică care conţine operatorii de relaţie cunoscuţi : =, <, <=, >, >=, < >.

Valoarea unei expresii de relaţie este o valoare logică de ADEVĂRAT sau FALS.

EXEMPLE de expresii de relaţie:

A=B
A < > B
A <= C >= D < > REF

Expresia logică este expresia numerică care conţine operatorii logici: NOT, AND, OR.

Operatorii logici într-o expresie cu 2 operanzi au următorul efect:

A AND B — dacă este îndeplinită condiţia A şi condiţia B se continuă pe ramura DA (sau ADEVĂRAT);
— dacă una din condiţiile A şi B nu este îndeplinită se continuă pe ramura NU (sau FALS).

A OR B — dacă este îndeplinită condiţia A şi condiţia B se continuă pe ramura DA;
— dacă este îndeplinită condiţia A, dar nu este îndeplinită condiţia B se continuă pe ramura DA;
— dacă nu este îndeplinită condiţia A, dar este îndeplinită condiţia B se continuă pe ramura DA;

— dacă nu este îndeplinită nici condiţia A nici condiţia B se continuă pe ramura NU.

NOT A — dacă nu este îndeplinită condiţia A se continuă pe ramura DA;

— dacă este îndeplinită condiţia A se continuă pe ramura NU.

2° Expresii alfanumerice

Expresiile alfanumerice sînt compuse din constante alfanumerice, variabile simple sau indexate de tip şir de caractere sau combinaţii ale acestora, separate prin operatori ca:

— operatori de concatenare;

— operatori de relaţie.

Operatorul de concatenare, marcat prin semnul „+", alipeşte două şiruri de caractere.

EXEMPLE de concatenare:

a) "ABC"+"DEFGH" produce şirul "ABCDEFGH"
b) "A1A2"+"100" produce şirul "A1A2100"

Operatorii de relaţie în cazul expresiilor alfanumerice operează asupra şirurilor de caractere cu respectarea ordinii alfabetice. Compararea a două şiruri de caractere se face caracter cu caracter de la stînga spre dreapta.

Operatorii de relaţie permişi sînt cei cunoscuţi : =; <; <=; >; >=; <>.

8.4.5. Funcţii

Funcţia în acepţiunea limbajului BASIC are aceeaşi semnificaţie ca în matematică: unui element dintr-o mulţime i se asociază un element din altă mulţime, respectîndu-se o anumită regulă.

Funcţiile sînt de 2 feluri, potrivit valorilor pe care le manipulează:

— funcţii matematice şi

— funcţii pe şir de caractere.

Funcţiile prezentate în continuare sînt funcţii standard adică au un mnemonic (nume) predefinit. Ele sînt apelate prin acest mnemonic (nume) şi se introduc într-o linie program printr-o singură acţionare a tastei pe care sînt specificate.

1° Funcţii matematice

Funcţia matematică este funcţia ce lucrează cu valori numerice şi corespunde unei funcţii matematice cunoscute.

Calculatorul HC-85 dispune de următoarele funcţii matematice:

SQR x rădăcina pătrată din x (radical din x, iar $x \geqslant 0$);
SIN x sinus de x (x în radiani);
COS x cosinus de x (x în radiani);
TAN x tangenta de x (x în radiani);

ASN	x	arcsinusul lui x;
ACS	x	arccosinusul lui x;
ATN	x	arctangenta a lui x;
EXP	x	exponenţiala lui x (e^x, unde e=2,71...);
LN	x	logaritmul natural din x (se poate utiliza la calculul unui logaritm în orice bază folosind formula LOGa x=LNx/LNa, iar x>0);
ABS	x	valoarea absolută a lui x (|x|);
INT	x	partea întreagă a lui x ([x]);
SGN	x	funcţia furnizează semnul lui x şi are valorile: 1 pentru x>0; 0 x=0; −1 x<0

2° Funcţii pe şir de caractere

Funcţia pe şir de caractere corespunde funcţiei care se aplică unui şir de caractere şi furnizează un număr sau invers, se aplică unui număr şi se obţine un şir de caractere.

Calculatorul HC-85 dispune de următoarele funcţii pe şir de caractere:

LEN	s$	se aplică şirului de caractere s$ şi generează un număr ce reprezintă lungimea şirului de caractere;
CODE	s$	se aplică şirului de caractere s$ şi întoarce codul primului caracter din şir (dacă şirul este vid, transmite 0);
VAL	s$	se aplică şirului de caractere s$ şi converteşte şirul în numere;
VAL $	s$	se aplică şirului de caractere s$ şi converteşte şirul în alt şir de caractere (este rar folosită);
CHR	n	se aplică numărului n şi generează caracterul ce are codul n;
STR	n	se aplică numărului n şi converteşte numărul în şir de caractere.

Capitolul 9

Instrucţiunile limbajului BASIC (tratare detaliată cu exemplificări)

Calculatorul poate fi folosit numai prin intermediul instrucţiunilor pentru care a fost proiectat.

O instrucţiune este alcătuiră dintr-un „cuvînt cheie" şi unul sau mai multe argumente:

<cuvînt cheie><argument 1>, <argument 2>. . .

Cuvîntul cheie este mnemonicul unui cuvînt din limba engleză ce defineşte numele instrucţiunii. Cuvîntul cheie este reprezentativ şi sugestiv.

Argumentul unei instrucţiuni este o constantă, o expresie, o condiţie, un mesaj, un şir de caractere, o funcţie sau o combinaţie a acestora. Toate tipurile de argumente trebuie să respecte regulile din capitolele precedente.

Argumentele unei instrucţiuni sînt delimitate de următorii separatori:
— virgulă;
— punct şi virgulă;
— apostrof.

Limbajul BASIC admite două tipuri de instrucţiuni:
— nenumerotate şi
— numerotate.

Instrucţiunea nenumerotată se numeşte şi *comandă*.

Comanda se execută imediat după apăsarea tastei CR (ultima tastă din rîndul 3), după care nu mai are nici un efect.

Instrucţiunea numerotată se numeşte simplu, *instrucţiune*.

Instrucţiunea nu se execută imediat, ci se stochează ca linie program şi se poate executa printr-o comandă RUN ori de cîte ori se doreşte acest lucru. *Tastarea lui CR, atît la terminarea unei comenzi cît şi la terminarea unei linii program este obligatorie.* Ea este manevra prin care datele tastate sînt încărcate în memorie şi poate fi asimilată cu o „poartă de intrare".

O linie program poate conţine una sau mai multe instrucţiuni; separarea instrucţiunilor dintr-o linie se face prin caracterul „:" (două puncte).

Numerele date liniilor trebuie să fie întregi şi cuprinse între 1 şi 9999.

Totalitatea liniilor program ce reprezintă transpunerea unui algoritm în vederea rezolvării unei probleme cu ajutorul calculatorului se numeşte *program*.

Listarea şi execuţia unui program se face în ordinea naturală a numerelor atribuite liniilor program componente. De aceea, este indicat ca la scrierea unui program să se numeroteze liniile din 10 în 10, dînd astfel posibilitatea inserării cu uşurinţă de linii noi.

Cursorul „>" indică linia curentă asupra căreia se pot face modificări sau după care se pot insera alte linii. De obicei, cursorul se află pe ultima linie introdusă, dar el poate fi deplasat în sus sau în jos prin apărarea simultană a tastei CAPS SHIFT şi a săgeţilor.

O instrucţiune nenumerotată se deosebeşte formal faţă de o instrucţiune numerotată prin inexistenţa numărului de linie. Deosebirea esenţială constă însă în modul de execuţie: prima se execută o singură dată şi nu mai poate fi reluată, a doua de cîte ori este nevoie.

Atît instrucţiunile nenumerotate cît şi instrucţiunile numerotate se supun unui grup comun de reguli, drept pentru care nu se va face nici o diferenţiere în prezentarea lor.

În cazul particular în care o anumită instrucţiune este utilizată sub formă de comandă se va specifica expres acest lucru.

S-a preferat abordarea instrucţiunilor în ordinea ce urmează, pentru a oferi posibilitatea de a se trece într-un timp cît mai scurt la conceperea de programe în limbajul de programare BASIC.

De asemenea, s-a ţinut permanent cont de alternarea prezentării instrucţiunilor simple sau atractive în raport cu prezentarea instrucţiunilor dificile, pentru a se doza efortul de înţelegere a limbajului BASIC. Ultimele instrucţiuni prezentate nu sînt cele mai neutilizate ci sînt cele care au sens într-o anumită dotare (monitor color, imprimantă).

Trebuie subliniat că exemplele alese pentru lămurirea şi folosirea instrucţiunilor sînt intenţionat simple. Ele scot în evidenţă utilizarea standard, urmărind familiarizarea eşalonată cu noul mod de a privi lucrurile prin intermediul unui limbaj de programare. Din loc în loc, acolo unde este oportun, se dau şi cîteva sfaturi practice generale, a căror respectare aduce un plus de uşurare în efortul de a stăpîni calculatorul în momentul elaborării unui program.

9.1. Instrucţiunea PRINT

Instrucţiunea PRINT este utilizată pentru editarea (scrierea) pe ecran a rezultatelor intermediare sau finale ale unei execuţii.

Formatul general este:

PRINT listă

unde „listă" este o constantă, o variabilă, o expresie, un şir de caractere separate prin virgulă, punct şi virgulă sau apostrof.

EXEMPLE:

1. Tipărirea numărului 12 se face prin comanda:

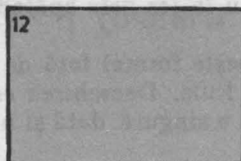
```
PRINT 12
```

2. Tipărirea şirului de caractere "SINT ELEV" se face prin comanda:

```
PRINT "SINT ELEV"
```

Trebuie reţinut că la terminarea introducerii fiecărei comenzi este obligatorie tastarea lui CR. Această manevră provoacă execuţia comenzii de către calculator.

Pe ecran apare:

1. 

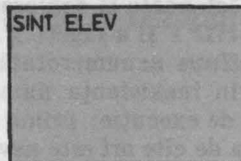
2. 

Fig. 9.1.

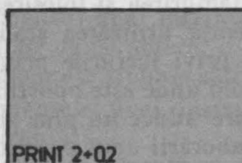
Cu ajutorul comenzii PRINT se fac calcule matematice imediate oricît de complicate ar fi, într-un timp record. Aceasta este una din modalitățile de efectuare a calculelor folosind calculatorul personal.

Limbajul BASIC oferă și alte posibilități pentru efectuarea de calcule, dar și pentru multe alte lucruri ce așteaptă numai să fie încercate și / sau descoperite.

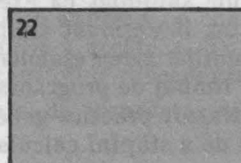
EXEMPLE de calcule imediate:

1. Adunarea dintre 2 și 0.2 se realizează prin comanda:

PRINT 2+0.2



Ecranul inainte de CR

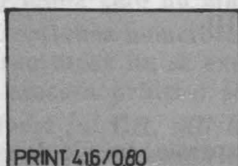


Ecranul după CR

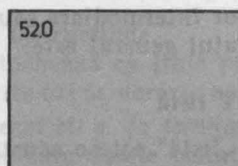
Fig. 9.2.

2. Împărțirea dintre 41.6 și 0.80 se realizează prin comanda:

PRINT 41.6/0.80



Ecranul înainte de CR

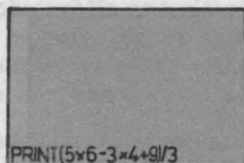


Ecranul după CR

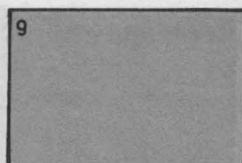
Fig. 9.3.

3. Calculul expresiei $\frac{5*6-3*4+9}{3}$ se realizează prin comanda:

PRINT (5 * 6—3 * 4+9)/3



Ecranul inainte de CR



Ecranul după CR

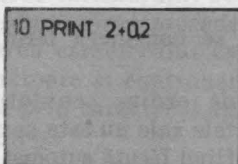
Fig. 9.4.

Comanda PRINT devine instrucţiunea PRINT dacă linia primeşte un număr de ordine şi în acest caz trebuie tastat CR la terminarea introducerii tuturor argumentelor instrucţiunii (dacă linia program conţine numai o singură instrucţiune). Altfel spus, tastarea lui CR marchează terminarea oricărei linii program, care este astfel memorată şi poate fi executată în urma unei comenzi RUN (execută).

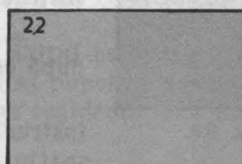
Calculele din exemplele anterioare se pot efectua şi prin instrucţiunea PRINT ajungîndu-se astfel la scrierea celor mai simple programe.

Program 1

```
10 PRINT 2 +0.2
   RUN
```



Ecranul după PRINT urmat de CR

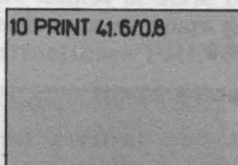


Ecranul după RUN urmat de CR

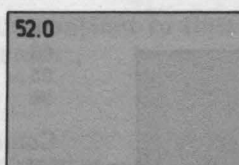
Fig. 9.5.

Program 2

```
10 PRINT 41.6/0.8
   RUN
```



Ecranul după PRINT urmat de CR



Ecranul după RUN urmat de CR

Fig. 9.6.

Program 3

```
10 PRINT (5 * 6—3 * 4+9)/3
   RUN
```

10 PRINT(5*6-3*4+9)/3

Ecranul după PRINT urmat de CR

9

Ecranul după RUN urmat de CR

Fig. 9.7.

Acestea sînt cele mai simple programe nu numai pentru că sînt formate dintr-o singură linie program ci şi pentru că şi-au propus spre rezolvare o singură problemă, şi ea simplă de fapt. De obicei, un program este alcătuit din mai multe linii program. De exemplu, toate calculele anterioare pot fi grupate într-un singur program de forma următoare:

```
10  PRINT  2+0.2
20  PRINT  41.6/0.8
30  PRINT  (5 * 6-3 * 4+9)/3
    RUN
```

Pe ecran rezultatele sînt acum afişate ca în figura 9.8.

22
52.0
9

Fig. 9.8.

Următoarele observaţii rămîn valabile pentru toate instrucţiunile:

— o linie program se consideră terminată numai după ce s-a tastat CR;

— între numărul de ordine, cuvîntul cheie al instrucţiunii şi argumentele sale nu este nevoie să se lase spaţiu (blanc), operaţia fiind făcută automat de caculator.

Programul anterior poate fi completat şi cu alte calcule, ca de exemplu împărţirea lui 7 la 2.

Pentru aceasta se introduce linia program:

```
25  PRINT  7/2
```

Respectînd convenţia de tastare a lui CR după fiecare linie program, se observă că ordonarea liniilor se face automat, iar noul program arată astfel:

```
10  PRINT  2 +0.2
20  PRINT  41.6/0.8
25  PRINT  7/2
30  PRINT  (5 * 6 -3 * 4 +9)/3
```

22
52.0
3.5
9

Fig. 9.9.

Comanda RUN urmată de tastarea lui CR face posibilă execuţia programului ce afişează pe ecran numerele din fig. 9.9.

În concluzie, orice program poate fi îmbogăţit cu alte linii program tastînd consecutiv, după ultima linie a lui, noile linii. Numărul de ordine al unei noi linii introduse poate fi mai mic sau mai mare decît ultimul număr de ordine, potrivit locului în care se doreşte inserarea. Ordonarea efectivă a liniilor program este făcută automat de către calculator.

Din observaţia anterioară se deduce modul prin care un program poate fi modificat pe parcursul exploatării sale. Pentru ca orice modificare să se facă cu un minim de efort, se recomandă ca numerele de ordine a liniilor unui program, în prima etapă a validării pe calculator, să respecte următoarele reguli:

— numerotarea să nu se înceapă de la 1 ci de la 10;
— numerotarea să nu se facă din 1 în 1 ci din 10 în 10.

O situaţie des întîlnită constă în renunţarea la o linie program.

O linie program este ştearsă prin simpla tastare a numărului său de ordine urmat imediat de tastarea lui CR.

Din programul:

```
10  PRINT  2 +0.2
20  PRINT  41.6/0.8
25  PRINT  7/2
30  PRINT  (5 * 6 -3 * 4 +9)/3
```

dispare împărţirea lui 41.6 cu 0.80 prin introducerea liniei:

```
20 <CR>
```

Între argumentele instrucţiunii PRINT se găsesc diverşi separatori. Folosirea acestora are următorul efect:

a) utilizarea caracterului virgulă determină începerea tipăririi fie pe marginea din stînga, fie în mijlocul ecranului, în funcţie de ce urmează după virgulă;

b) utilizarea caracterului punct şi virgulă determină tipărirea şirului imediat următor după şirul precedent;

c) utilizarea caracterului apostrof determină saltul cursorului la începutul liniei următoare şi continuarea tipăririi din acel punct ca şi cum elementele despărţite prin caracterul apostrof ar fi fost sub incidenţa unor instrucţiuni PRINT succesive.

Pentru ca instrucţiunea PRINT să nu determine saltul la linia următoare este necesar ca instrucţiunea PRINT precedentă să se termine cu caracterele virgulă sau punct şi virgulă.

Se constată că varianta cu virgulă împarte totul în două coloane, cea cu punct şi virgulă scrie totul compact, iar cea cu apostrof scrie un număr pe o linie.

Dacă după argumentele lui PRINT nu se pune nici un semn de punctuaţie atunci fiecare număr se scrie pe o altă linie.

Dacă se doreşte să apară pe ecran o linie fără conţinut (o linie liberă) se foloseşte instrucţiunea PRINT fără nici un argument.

```
HC 85 are 5 moduri de lucru: 1-K,
2-L, 3-C, 4-E, 5-G
```

Fig. 9.10.

EXEMPLE:

Program 1

```
10  PRINT "HC-85 are 5 moduri de lucru:";
20  PRINT "1—K, 2—L, 3—C, 4—E, 5—G"
```

afişează pe ecran rezultatele ca în fig. 9.10.

Program 2

```
10  PRINT "HC-85 are 5 moduri de lucru:";
10  PRINT "1-K,";
30  PRINT "2-L,";
40  PRINT "3-C,";
```

```
50    PRINT "4-E,";
60    PRINT "5-G";
```

afişează pe ecran rezultatele ca şi programul 1.

Program 3

Programul 3 este conceput pe structura programului 2 modificînd liniile 10 –60 prin ştergerea caracterului virgula şi a caracterului punct şi virgulă.

```
10    PRINT "HC-85 are 5 moduri de lucru:"
20    PRINT "1-K"
30    PRINT "2-L"
40    PRINT "3-C"
50    PRINT "4-E"
60    PRINT "5-G"
```

Pe ecran rezultatele programului 3 sînt afişate ca în fig. 9.11

```
HC-85 are 5 moduri de lucru:
1-K
2-L
3-C
4-E
5-G
```

Fig. 9.11.

Program 4

```
10    PRINT "HC-85 are 5 moduri de lucru:"
20    PRINT "1-K"'
30    PRINT "2-L"'
40    PRINT "3-C"'
50    PRINT "4-E"'
60    PRINT "5-G"'
```

afişează pe ecran rezultatele ca şi programul 3 deşi liniile 20 –60 sînt modificate faţă de liniile programului 3 prin adăugarea la extremitatea dreaptă a caracterului apostrof.

```
HC-85 are 5 moduri de lucru:
1-K           2-L
3-C           4-E
5-G
```

Fig. 9.12.

Program 5

```
10    PRINT "HC-85 are 5 moduri de lucru:"
20    PRINT "1-K",
30    PRINT "2-L",
40    PRINT "3-C",
50    PRINT "4-E",
60    PRINT    "5-G"
```

afişează pe ecran rezultatele ca în fig. 9.12.

```
HC-85 are 5 moduri de lucru:
1-K           2-L
3-C    4-E    5-G
```

Fig. 9.13.

Program 6

Programul 6 este identic cu programul 5 cu excepţia liniei 40 din care dispare virgula.

```
10    PRINT "HC-35 are 5 moduri de lucru:"
20    PRINT "1-K",
30    PRINT "2-L",
40    PRINT "3-C"
50    PRINT "4-E",
60    PRINT "5-G"
```

Pe ecran rezultatele programului 6 sînt afişate ca în fig. 9.13.

```
HC-85 are 5 moduri de lucru:
1-K          2-L
3-C
4-E          5-G
```

Fig. 9.14.

Program 7

Programul 7 este programul 6 cu linia 40 modificată prin adăugare la sfîrşit a semnului punct şi virgulă.

```
10   PRINT "HC-85 are 5 moduri de lucru:"
20   PRINT "1-K",
30   PRINT "2-L",
40   PRINT "3-C";
50   PRINT "4-E",
60   PRINT "5-G"
```

Pe ecran rezultatele programului 7 sînt afişate ca în fig. 9.14.

```
HC-85 are 5 moduri de lucru :
1-K
2-L          3-C 4-E
5-G
```

Fig. 9.15.

Program 8

```
10   PRINT "HC-85 au 5 moduri de lucru:"
20   PRINT "1-K"'
30   PRINT "2-L",
40   PRINT "3-C";
50   PRINT "4-E",
60   PRINT "5-G"
```

afişează pe ecran rezultatele ca în fig. 9.25

9.2. Instrucţiunea RUN

Din exemplele anterioare se observă că pentru execuţia unui program trebuie utilizată comanda RUN.

RUN este des folosită sub forma de comandă.

Formatul general este:

RUN nr. linie

unde

a) dacă „nr. linie" este specificat, execuţia programului cuprinde liniile începînd de la numărul de ordine „nr. linie" pînă la ultima linie program introdusă;

b) dacă „nr. linie" nu este specificat execuţia programului cuprinde în întregime liniile introduse.

Trebuie reţinut că RUN poate fi foarte bine şi instrucţiune. Programul care conţine o instrucţiune RUN se execută numai după un RUN, care de această dată este comandă. Execuţia unui astfel de program durează la nesfîrşit. Oprirea se realizează prin tastarea lui BREAK, care înseamnă „întrerupe" şi se obţine prin acţionarea simultană a tastelor CS şi SPACE.

EXEMPLU:

Programul

```
10   PRINT "a"
```

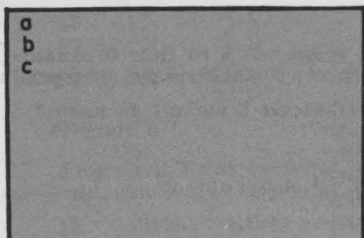


Fig. 9.16.

```
20   PRINT "b"
30   PRINT "c"
```

afişează pe verticală grupul de litere a, b, c ca în fig. 9.16.

Prin completarea programului anterior cu linia

```
40 RUN
```

se obţine afişarea continuă a grupului de litere a, b, c (ca o pîlpîire), iar oprirea execuţiei se realizează prin tastarea lui BREAK.

Pe ecran apare mesajul:

L   BREAK   into program, 40: 1

Dacă se doreşte o nouă afişare, se tastează RUN urmat de CR şi se efectuează o nouă execuţie a programului.

Execuţia unui program, fie că are sau nu o linie program ce-l conţine pe RUN, se obţine prin comanda RUN.

9.3. Instrucţiunea DELETE

De cîte ori este nevoie să se şteargă un simbol se utilizează DELETE. Ea este folosită sub formă de comandă.

Formatul general este:

DELETE

Se impun următoarele precizări:

a) dacă se tastează DELETE (acţionarea simultană a tastelor CS şi 0) o singură dată atunci se produce ştergerea ultimului simbol introdus;

b) dacă se tastează DELETE de mai multe ori atunci se produce ştergerea unui număr de simboluri egal cu numărul tastărilor efectuate, începînd cu ultimul simbol introdus;

c) introducerea de la tastatură a comenzii DELETE nu se vizualizează pe ecran sau altfel spus numele comenzii DELETE nu se afişează pe ecran ci numai efectul ei (ştergerea efectivă şi pe rînd a simbolurilor de la dreapta la stînga);

d) cu DELETE se efectuează ştergerea unor simboluri în cadrul liniei program curente (linia pentru care nu s-a tastat încă CR);

e) în cazul în care se doreşte să se renunţe (din anumite considerente) la o linie program deja introdusă se va folosi procedeul de suprapunere al unei linii vide peste respectiva linie (linia formată numai din numărul de ordine al liniei ce trebuie eliminată, terminată brusc prin tastarea lui CR) şi nu se va recurge la utilizarea lui DELETE.

9.4. Instrucţiunea GO TO

Instrucţiunea GO TO serveşte la modificarea ordinei secvenţei de execuţie a liniilor unui program. Această instrucţiune realizează un salt necondiţionat la numărul de linie specificat.

Formatul general este:

GO TO nr linie

unde „nr. linie" este numărul liniei program de la care se va începe reluarea.

Modificarea programului de la pct. 9.2 prin înlocuirea liniei 40 cu o nouă linie:

40 GO TO 10

face ca programul să se reia mereu de la început. Ca rezultat ecranul se umple pe verticală cu literele a, b şi c.

În momentul cînd ecranul s-a umplut în partea de jos a lui apare mesajul:

scroll ?

(în limba engleză scroll înseamnă „sul de hîrtie").

Dacă se apasă pe orice tastă înseamnă că se doreşte „scroll", echivalent cu continuarea afişării.

În acest moment ultimul rînd tipărit ia locul penultimului rînd, penultimul rînd ia locul antepenultimului rînd ş.a.m.d, astfel că primul rînd tipărit iese din ecran, iar pe ultimul rînd de acum (rămas liber) se poate tipări.

Înlocuirea unui rînd cu alt rînd, de jos în sus, se produce rapid şi rularea afişării şirului de numere seamănă cu o „defilare". Cînd ecranul s-a umplut, apare din nou mesajul:

scroll ?

Dacă se apasă pe tasta N (N de la nu) apare mesajul:

D BREAK — CONT repeats

şi afişarea este oprită.

De fapt răspunzîndu-se prin „nu" s-a produs şi oprirea execuţiei programului.

Trebuie specificat clar că, GO TO are efect de lansare în execuţie a unui program (aşa cum se vede de fapt şi din exemplul ales) la fel ca RUN. Există însă între RUN şi GO TO o deosebire esenţială:

— RUN şterge valorile tuturor variabilelor rămase dintr-o execuţie anterioară;

— GO TO păstrează (nu schimbă) aceste valori.

9.5. Instrucţiunea CLS

Instrucţiunea CLS provoacă ştergerea ecranului.

Formatul general este:

CLS

(fără nici un argument).

Programul de la pct 9.4 diferă de programul de la pct. 9.2 prin conţinutul liniei 40. Cu ajutorul instrucţiunii CLS programul de la pct. 9.4 poate avea acelaşi efect cu cel de la pct. 9.2. prin:

— introducerea unei noi linii

```
5 CLS
```

— modificarea liniei 40

```
40 GO TO 5
```

Programul arată acum astfel:

```
5   CLS
10  PRINT "a"
20  PRINT "b"
30  PRINT "c"
40  GO TO 5
```

şi afişează continuu pe verticală grupul de litere a, b şi c.

Ştergerea ecranului se realizează şi cu ajutorul comenzilor RUN şi CLEAR (care au şi alte funcţii), dar numai instrucţiunea CLS poate şterge ecranul în anumite momente ale execuţiei unui program.

Instrucţiunea CLS şterge ecranul, dar nu şterge şi programul din memoria calculatorului.

9.6. Instrucţiunea NEW

Instrucţiunea NEW şterge din memorie programul curent pentru a face posibilă introducerea unui nou program.

NEW este folosită în majoritatea cazurilor sub formă de comandă.

Formatul general este:

NEW

(fără nici un argument).

9.7. Instrucţiunea LIST

Instrucţiunea LIST serveşte la extragerea pe ecranul televizorului a unei părţi sau a întregului program curent (existent în momentul acela în memoria internă).

LIST este des folosită sub formă de comandă.

Formatul general este:

LIST nr. linie

unde

a) dacă „nr. linie" este specificat, se produce listarea programului de la linia program cu acel număr de ordine pînă la linia curentă (ultima linie program introdusă) care poate fi şi sfîrşitul programului;

b) dacă „nr. linie" nu este specificat, se produce listarea programului de la început (prima linie program introdusă) pînă la linia curentă.

De reţinut este faptul că linia curentă (cea cu >) apare întotdeauna pe ecran, în mod normal în poziţie centrală. Calculatorul memorează numărul liniei curente precum şi numărul primei linii din partea de sus a ecranului.

Cînd încearcă să listeze, primul lucru pe care-l face este să compare prima linie de pe ecran cu linia curentă. Dacă prima linie de pe ecran este mai mare decît linia curentă, atunci cursorul va apare pe prima linie a ecranului. Altfel, listarea constă în tipărirea pe ecran în mod defilare a programului cuprins între prima linie şi linia curentă.

Oricum, mai întîi se efectuează un calcul aproximativ pentru a vedea cît timp ia listarea şi dacă aceasta este prea lungă, linia din vîrf se mută mai jos pentru a fi mai aproape de linia curentă. Acum avînd stabilită linia din vîrf, listarea poate începe. Dacă linia curentă a fost listată, listarea se opreşte cînd s-a ajuns la sfîrşitul programului sau în partea de jos a ecranului.

Listarea unui program, după execuţia sa, se face şi prin simpla tastare a lui CR.

9.8. Instrucţiunea PRINT AT

Ecranul este împărţit în 24 linii de afişare, fiecare cu 32 de caractere.

Ecranul are două părţi. Partea de sus de 22 linii este folosită pentru listarea liniilor programului sau a rezultatelor. Cele două linii de jos se folosesc pentru introduceri curente de date.

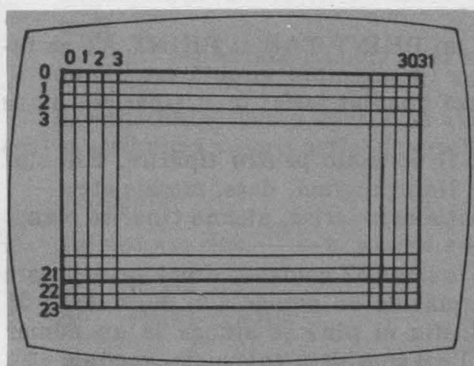


Fig. 9.17.

În acest fel, ecranul are disponibile pentru utilizator: 22 de linii şi 32 coloane. Liniile sînt numerotate de la 0 la 21 (de sus în jos), iar coloanele de la 0 la 31 (de la stînga spre dreapta) ca în fig 9.17.

Instrucţiunea PRINT AT deplasează cursorul (locul în care se va tipări) în linia şi coloana specificată şi tipăreşte lista de caractere dorită.

Formatul general este:

PRINT AT linie, coloană; listă

unde

— „linie" este numărul liniei unde se doreşte tipărirea;

— „coloană" este numărul coloanei unde se doreşte tipărirea;

— „listă" reprezintă şirul de caractere ce urmează a fi tipărit în locul stabilit.

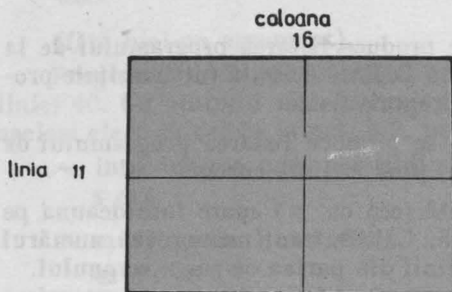


Fig. 9.18.

EXEMPLU:

PRINT AT 11, 16; " . "
imprimă un punct în centrul ecranului (fig. 9.18).

9.9. Instrucţiunea PRINT TAB

Instrucţiunea PRINT TAB deplasează cursorul în cadrul aceleiaşi linii, în coloana specificată şi tipăreşte lista de caractere dorită. Cursorul se deplasează pe linia următoare în cazul excepţie, cînd poziţia de tipărire specificată se află înaintea poziţiei de tipărire actuală.

Formatul general este:

PRINT TAB coloană; listă

unde

— „coloană“ este numărul coloanei din cadrul liniei curente unde se dorește tipărirea;

— „listă“ reprezintă şirul de caractere ce urmează a fi tipărit în locul stabilit.

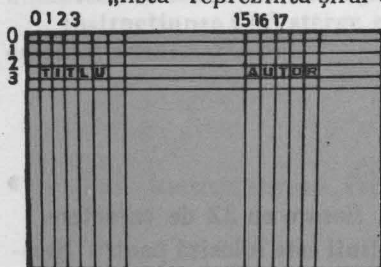


Fig. 9.19.

EXEMPLU:

Programul

```
10   PRINT AT 3,1; "TITLU"
20   PRINT TAB 16; "AUTOR"
```

tipăreşte pe linia 3, coloana 1 textul TITLU şi pe aceeaşi linie, coloana 16 textul AUTOR (fig. 9.19)

De reţinut următoarele:

a) după introducerea argumentelor lui PRINT TAB şi PRINT AT se folosește „;“. Dacă se folosește alt separator (de exemplu virgulă sau blanc), cursorul se poziţionează, dar se deplasează imediat astfel încît tipărirea nu se execută acolo unde s-a dorit;

b) liniile ecranului 22 şi 23 nu pot fi utilizate pentru tipărire. Ele sînt rezervate pentru introducere de comenzi, linii program, date, mesaje etc;

c) tipărind cu PRINT AT într-o poziţie deja scrisă, ultima tipărire o anulează pe precedenta;

d) calculatorul consideră că există numai 32 coloane, drept pentru care stabilirea coloanei unui PRINT TAB urmat de un număr mai mare decît 31 se face prin scăderea succesivă a lui 32 din el pînă se ajunge la un număr cuprins între 0 şi 31. Se spune că PRINT TAB consideră coloanele „modulo 32“.

EXEMPLU:

PRINT TAB 32; 1	**este echivalent cu**	**PRINT TAB 0; 1**
PRINT TAB 75; "$"		**PRINT TAB 11; "$"**

fiindcă 32—32 = 0 şi
75—32 = 43; 43—32 = 11

9.10. Instrucţiunea REM

Instrucţiunea REM introduce un mesaj într-un program. Mesajul este un comentariu la adresa programului sau la adresa unui grup de linii, în vederea scoaterii în evidenţă a unor caracteristici. Un program bine comentat este mai uşor de depanat şi înţeles, drept pentru care îl pot adapta şi folosi mai mulţi utilizatori.

Formatul general este:

REM listă

unde „listă" este un şir de caractere ce aduce unele lămuriri despre program.

Instrucţiunea REM nu este luată în considerare în nici un fel cînd se trece la execuţia programului.

EXEMPLU:

Programul de la pct. 9.2 poate primi numele „Afisează litere" prin completarea cu următoarea linie program:

```
5 REM Afisează litere
```

În exemplu, numele dat programului prin introducerea liniei program 5 reprezintă o caracteristică ce lămureşte de fapt ceea ce şi-a propus să execute acest mic program.

9.11. Instrucţiunea LET

Instrucţiunea LET permite atribuirea de valori unei variabile.

Formatul general este:

LET var = expr.

unde

— „var" reprezintă numele unei variabile simple sau indexate;

— „expr" reprezintă o expresie numerică sau alfanumerică.

Efectul instrucţiunii constă în evaluarea expresiei din dreapta semnului egal şi atribuirea valorii astfel determinate variabilei din stînga semnului egal.

EXEMPLU:

Calculul expresiei $\frac{x+1}{2}$ în punctul x=1.

```
10   LET x=1
20   PRINT (x+1)/2
```

EXEMPLU:

Concatenarea şirurilor de caractere "Ionescu" şi "Ana"

```
10   LET N$ = "Ionescu"
20   LET P$ = "Ana"
30   LET S$ = N$+P$
40   PRINT S$
```

Şirul S tipărit prin linia 40 are valoarea „Ionescu Ana".

Se observă din exemplul dat că un şir de caractere poate fi atribuit ca valoare unei variabile şir sau poate fi tipărit cu o instrucţiune PRINT.

Fiind dat un şir, un subşir al lui se formează din cîteva caractere consecutive ale lui, luate în secvenţă (unul după altul).

EXEMPLU:

Subşirurile
"eu"
"sint"
"elev"
"eu sint"
"sint elev"
sint subşiruri ale şirului
"eu sint elev"

Subşirul
"eu elev"
nu este un subşir al şirului
"eu sint elev"
pentru că s-a sărit peste "sint".

Subşirul
"eu   sint"
nu este subşir al şirului
"eu sint elev"
pentru că între "eu" şi "sint" s-au introdus mai multe blancuri (spaţii), blancul fiind şi el considerat un caracter distinct.

Manipularea subşirurilor în BASIC se face sub forma următoare:

s (n1 TO n2)

unde

— „s" este un şir de caractere sau o variabilă şir;

— „n1", „n2" sînt numere întregi nenegative ce reprezintă numărul de ordine al caracterului de început, respectiv de sfîrşit, din subşir.

Dacă n1 > n2 rezultatul este şirul vid ("").

Dacă nu se precizează începutul subşirului, n1 se ia implicit egal cu 1.

Dacă nu se precizează sfîrşitul subşirului, n2 se ia implicit egal cu lungimea şirului.

EXEMPLE de manipulări de subşiruri de caractere:

```
"abcdef" (2 TO 5) = "bcde"
"abcdef" (TO 5) = "abcdef" (1 TO 5) = "abcde"
"abcdef" (2 TO) = "abcdef" (2 TO 6) = "bcdef"
"abcdef" (TO) = "abcdef" (1 TO 6) = "abcdef"
"abcdef" (3) = "abcdef" (3 TO 3) = "c"
"abcdef" (5 TO 7) dă mesaj de eroare deoarece şirul are numai şase caractere.
"abcdef" (8 TO 7) = ""
"abcdef" (1 TO 0) = ""
```

EXEMPLU:

Prin programul

```
10    LET a$ = "123456"
20    LET a$ (3 TO 4) = "AB"
30    PRINT a$
```

şirul de caractere „123456" este transformat în şirul de caractere "12AB56".

9.12. Instrucţiunea BEEP

Calculatorul produce sunete cu ajutorul instrucţiunii BEEP. Formatul general este:

BEEP d, i

unde

— „d" este o expresie numerică ce indică durata în secunde a sunetului respectiv;

— „i" este o expresie numerică ce reprezintă înălţimea sunetului măsurat în semitonuri relativ la DO central (este admisă şi o valoare negativă).

Pentru a transcrie muzică este indicat să se scrie pe marginea fiecărui spaţiu şi linii a portativului înălţimea corespunzătoare, ţinînd cont de armura cheii.

Exemplu:

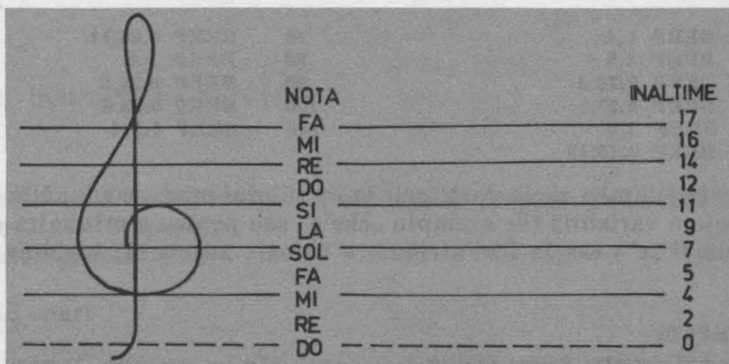


Fig. 9.20.

Trebuie hotărîtă în prealabil şi durata notelor, după cum se observă în fig. 9.21.

la tipul de notă corespunde durata în secunde



Fig. 9.21.

După ce s-au stabilit duratele şi înălţimile notelor, oricărei partituri i se ataşează un set de numere ce va fi utilizat la reproducerea ei de calculator.

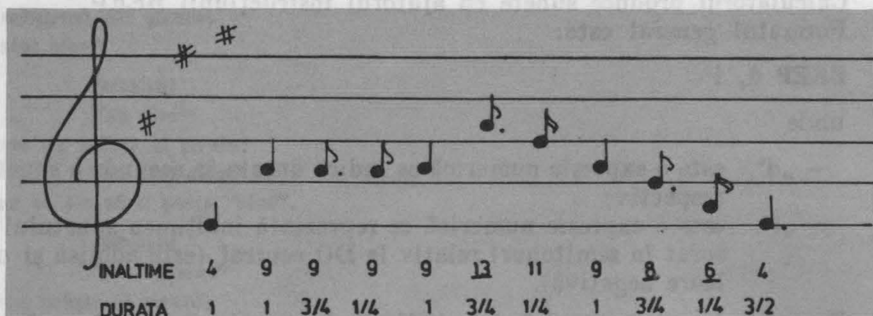


Fig. 9.22.

Notele DO, FA, SOL prevăzute cu # (diez) au înălţimea cu o unitate mai mare decît notele obişnuite.

Şi acum programul ataşat partiturii din figura 9.22

```
10   BEEP 1,4              70   BEEP 0.25,11
20   BEEP 1,9              80   BEEP 1,9
30   BEEP 0.75,9           90   BEEP 0.75,8
40   BEEP 0.25,9          100   BEEP 0.25,6
50   BEEP 1,9             100   BEEP 1.5,4
60   BEEP 0.75,13
```

Se poate schimba cheia partiturii în următorul mod: se adună la înălţimea fiecărei note o variabilă (de exemplu „cheie" sau pentru a minimiza efortul de tastare numai „c") căreia i se atribuie o valoare adecvată, înaintea execuţiei piesei.

EXEMPLU:

Programul anterior devine atunci:

```
10   BEEP 1, c +4
20   BEEP 1, c +9
30   BEEP .75, c +9
40   BEEP .25, c +9
50   BEEP 1, c +9
60   BEEP .75, c +13
70   BEEP .25, c +11
```

```
80   BEEP 1,c +9
90   BEEP .75, c +8
100  BEEP .25, c +6
110  BEEP 1.5, c +4
```

Variabila „c“ poate lua diferite valori ca: 0 pentru DO minor, 2 pentru RE minor, 12 pentru DO minor octava superioară etc. (se pot folosi şi valori zecimale).

După stabilirea valorii lui „c“ este absolut necesar completarea programului cu linia 5:

```
5 LET c =2
```

Folosind procedeul descris, se poate realiza acordarea calculatorului cu un instrument.

De asemenea, este posibil să se execute piese cu viteze diferite. În exemplul ales „o pătrime“ a fost programată să dureze un sfert de secundă. Dacă se introduce o variabilă („pătrime“ sau „p“ analog cu „cheie“ sau „c“) căreia i se atribuie o anumită valoare (de exemplu 0.5), programul se transformă astfel:

```
5    LET  c =2: LET p =.5
10   BEEP p +1, c +4
20   BEEP p +1, c +9
30   BEEP p +.75, c +9
40   BEEP p +.25, c +9
50   BEEP p +1, c +9
60   BEEP p +.75, c +13
70   BEEP p +.25, c +11
80   BEEP p +1, c +9
90   BEEP p +.75, c +8
100  BEEP p +.25, c +6
110  BEEP p +1.5, c +4
```

Prin schimbarea valorilor variabilelor cu numele „cheie“ sau „pătrime“ este posibilă execuţia aceluiaşi program în numeroase moduri, epuizînd toate dorinţele (preferinţele) definite prealabil.

9.13. Instrucţiunea PAUSE

Instrucţiunea PAUSE realizează o pauză în program în timpul căreia nu se desfăşoară nici o operaţie.

Formatul general este:

PAUSE const

unde „const“ este un număr între 0 şi 65535.

Numărul $65535=2^{16}-1$ şi este determinat de mărimea memoriei calculatorului, respectiv 64 KO ($64=2^6$ şi $KO=1024=2^{10}$).

Instrucţiunea PAUSE opreşte execuţia programului menţinînd activ ecranul pe durata specificată în perioade de baleiaj.

O perioadă de baleiaj este echivalentă cu 20 ms.

Pauza maximă posibilă corespunde instrucţiunii

PAUSE 65535

şi înseamnă aproximativ 22 minute.

Pauza minimă posibilă corespunde instrucţiunii

PAUSE 0

care în realitate înseamnă oprirea definitivă a execuţiei programului.

O pauză obţinută folosind PAUSE poate fi scurtată apăsînd orice tastă (cu excepţia lui SPACE şi CAPS SHIFT, taste ce produc BREAK).

EXEMPLU:

Se tipăresc pe cîte un rînd fiecare, 5 numere, cu o pauză de timp între ele de aproximativ 1, 2, 3, 4 secunde.

```
10   PRINT 111
20   PAUSE 50
30   PRINT 222
40   PAUSE 100
50   PRINT 333
60   PAUSE 150
70   PRINT 444
80   PAUSE 200
90   PRINT 555
```

Cu instrucţiunea PAUSE se pot reproduce şi pauzele muzicale.

EXEMPLU:

```
10   BEEP  .75,4
20   BEEP  .25,2
30   BEEP  .5,1
40   BEEP  .5,2
50   BEEP  .5,4
60   BEEP  .5,6
70   BEEP  .5,7
80   PAUSE 20
90   BEEP  .75,4
100  BEEP  25,6
110  BEEP  .5,7
120  BEEP  .5,7
130  BEEP  .5,4
140  BEEP  .5,2
150  BEEP  1,1
```

9.14. Instrucţiunea LOAD

Calculatorul HC-85 are posibilitatea să încarce în memorie ceea ce se găseşte pe bandă magnetică. Conectarea calculatorului la casetofon se face cu ajutorul unui cablu special.

Instrucţiunea LOAD execută încărcarea.

LOAD este folosită sub forma de comandă.

a) Dacă se doreşte încărcarea unui program de pe casetă în memorie se foloseşte comanda LOAD cu formatul:

LOAD "nume"

unde „nume" este şirul de caractere ce defineşte numele programului ce se doreşte încărcat.

Cînd se tastează LOAD, înainte de a se încărca noul program se şterge vechiul program împreună cu toate variabilele lui.

Comanda LOAD fără un nume specificat LOAD " " încarcă primul program găsit pe casetă.

b) Dacă se doreşte încărcarea unor şiruri de caractere de pe casetă în memorie se foloseşte LOAD cu formatul:

LOAD "nume" DATA şir ( )

unde
— "nume" este numele sub care se găseşte înregistrat şirul pe bandă;
— „şir ( )" este noul nume sub care se va încărca şirul în memorie.

Exemplu:
Comanda

LOAD "test" DATA T ()

caută pe bandă şirul cu numele "test", cînd îl găseşte verifică dacă nu mai există în memorie un şir T, îl anulează şi încarcă şirul „test" în memoria calculatorului sub numele de T.

9.15. Instrucţiunea SAVE

Calculatorul HC-85 are posibilitatea să înregistreze pe bandă magnetică părţi din memorie, operaţiune care se numeşte „salvarea" conţinutului respectivei părţi din memorie.

Ea se execută cu ajutorul instrucţiunii SAVE.

SAVE este folosită sub formă de comandă.

a) Dacă se doreşte salvarea unui program pe casetă se foloseşte comanda SAVE cu formatul:

SAVE "nume"

unde "nume" este şirul de caractere ce defineşte numele sub care va fi salvat programul.

Numele programului este compus din maximum 10 caractere litere — şi/sau cifre — şi este fixat de utilizator.

Cînd este gata de salvare calculatorul afişează mesajul:

Start tape then press any key

care înseamnă „porneşte casetofonul şi apoi apasă orice tastă".

La terminarea înregistrării apare mesajul:

O.K.

adică „foarte bine".

Este posibil să se înregistreze un program pe casetă astfel încît atunci cînd este reîncărcat în memorie să se lanseze automat de la o linie anume. Instrucţiunea are atunci formatul:

SAVE şir LINE număr

şi face ca programul încărcat cu LOAD (dar nu şi cu MERGE — vezi 9.17) să fie rulat automat de la linia specificată prin „număr". Dacă nu loc suficient în memorie programul vechi precum şi vechile variabile nu sînt şterse şi apare un mesaj de eroare:

Out of memory

adică „depăşire de memorie".

b) Dacă se doreşte salvarea unui şir de caractere pe casetă se foloseşte comanda SAVE cu formatul:

SAVE "nume" DATA şir ( )

unde

— "nume" este noul nume sub care va fi înregistrat şirul pe bandă;

— „şir ( )" este numele şirului din memorie ce va fi salvat.

EXEMPLU:

Comanda

SAVE "NOU" DATA a ( )

înregistrează şirul a ( ) pe banda magnetică sub numele "NOU".

9.16. Instrucţiunea VERIFY

După înregistrarea pe banda magnetică este bine să se facă verificarea corectitudinii înregistrării.

Pentru verificare se reglează volumul casetofonului la nivel mediu şi se poziţionează banda în punctul în care a început înregistrarea.

Instrucţiunea VERIFY execută verificarea corectitudinii înregistrării. VERIFY este folosită sub formă de comandă.

a) Dacă se doreşte verificarea salvării unui program se foloseşte comanda VERIFY cu formatul:

VERIFY "nume"

unde "nume" este numele programului ce a fost înregistrat pe bandă şi pentru care se verifică corectitudinea înregistrării.

Comanda VERIFY verifică dacă programul şi variabilele înregistrate pe casetă sînt identice cu cele din memoria calculatorului. Dacă programul a fost înregistrat şi chemat corect, pe ecran apare:

Program "nume"

În timpul căutării programului specificat, calculatorul tipăreşte numele tuturor programelor pe care le întîlneşte, iar la sfîrşit apare mesajul:

OK

În cazul unei erori de înregistrare (eroare ce apare la VERIFY) se afişează mesajul:

R tape loading error

care înseamnă „eroare de înregistrare pe casetofon" şi se încearcă o nouă înregistrare.

b) Dacă se doreşte verificarea salvării unui şir de caractere se foloseşte comanda VERIFY cu formatul:

VERIFY "nume" DATA şir ( )

unde

— "nume" este numele sub care şirul a fost salvat;

— „şir ( )" este numele şirului din memorie în raport cu care se verifică corectitudinea salvării.

EXEMPLU:

VERIFY "NOU" DATA a ( )

verifică înregistrarea făcută pe casetă sub numele de "NOU" a şirului a ( ) din memorie.

9.17. Instrucţiunea MERGE

Instrucţiunea MERGE serveşte la încărcarea şi intercalarea unui program de pe bandă magnetică cu programul ce se află în memorie în acel moment.

MERGE este folosită sub formă de comandă.

Formatul general este:

MERGE "nume"

unde "nume" este numele programului de pe bandă ce urmează a fi încărcat şi intercalat cu programul din memorie.

MERGE este util în cazul testării unui program complex. În momente diferite se păstrează pe bandă diferite variante ale programului, ce vor fi intercalate între ele pînă se va ajunge la varianta finală (cea mai bună).

Comanda MERGE nu poate fi folosită la intercalarea şirurilor de caractere.

9.18. Instrucţiunea INPUT

Instrucţiunea INPUT oferă o modalitate de introducere de valori direct de la tastatură.

Formatul general este:

INPUT listă; var 1, var 2, ...

unde

— „listă" este o constantă şir de caractere ce punctează acţiunea de introducere de date şi reprezintă un mesaj (un comentariu);

— „var 1", sau „var 2" ...... este numele unei variabile simple sau indexate, numerice sau şir de caractere ce va primi valoarea tastată.

Argumentul „listă" nu este obligatoriu.

Atribuirea valorilor în urma unui INPUT se consideră terminată în momentul tastării lui CR.

EXEMPLU:

Programul 1 calculează şi tipăreşte perimetrul oricărui pătrat de latură L.

```
10 INPUT  L
20 PRINT  P=4 * L
30 GO TO  10
```

Imediat după comanda RUN calculatorul aşteaptă introducerea unei valori:

— dacă nu se tastează nimic, în partea de jos a ecranului apare un semn de întrebare şi calculatorul aşteaptă;

— dacă se şterge semnul de întrebare printr-un DELETE şi se introduce valoarea 2(L=2), sus pe ecran apare 8 (de la 4×2=8).

Prin linia 30 programul se reia automat şi introducîndu-se alte valori pentru L se obţin şi alte valori pentru P.

EXEMPLU:

Programul 1 poate fi modificat astfel:

```
10 INPUT "latura ="; L
20 PRINT "perimetrul ="; P=4 * L
30 GO TO 10
```

În această situaţie, în partea de jos a ecranului apare mesajul "latura=" care anunţă că se aşteaptă introducerea valorii ce va fi asignată laturii unui pătrat, iar la apariţia mesajului "perimetrul=" se anunţă că rezultatul corespunde perimetrului pătratului cu latura stabilită, după care urmează o nouă apariţie a mesajului "latura=" şi se continuă procesul de calcul.

O linie cu INPUT poate conţine o serie de separatori (punct şi virgulă, virgulă, apostrof) care au acelaşi efect ca într-o linie cu PRINT.

INPUT consideră orice element care începe cu o literă ca pe o variabilă asignabilă (căreia urmează să i se introducă valoarea de la tastatură). Instrucţiunea INPUT poate tipări şi mesaje; pentru a tipări un şir de caractere este suficientă introducerea acestuia între ghilimele. Dacă conţine şi valori de variabile, mesajul se închide între paranteze.

Dacă se doreşte citirea unei variabile de tip şir de caractere, a$, pe ecran apare caracterul ghilimele. Dacă această variabilă trebuie să ia valoarea unei alte variabile de tip şir definită în program, b$, aceasta se face prin ştergerea ghilimelelor şi introducerea numelui variabilei (b$).

EXEMPLU:

Programul calculează şi tipăreşte perimetrul unui dreptunghi cînd lungimea şi lăţimea se introduc de la tastatură.

```
10 INPUT "lungime ="; L; "latime ="; l
20 PRINT "perimetrul ="; p=2 * L+2 * l
30 GO TO 10
```

Mesajele "lungime=" şi "latime=" apar pe aceeaşi linie, unul după altul, imediat după ce s-a introdus primul număr.

Dacă se modifică linia 10 aşa fel încît după L să urmeze simbolul virgulă şi nu simbolul punct şi virgulă

10 INPUT "lungime=" ; L, "laţime="; l

mesajele "lungime=" şi "lăţime=" apar pe aceeaşi linie, primul la începutul rîndului, iar după ce se introduce o valoare, cel de-al doilea mesaj la mijlocul aceluiaşi rînd.

Dacă în linia 10 se înlocuieşte simbolul punct şi virgulă existent după L cu simbolul apostrof

10 INPUT "lungime="; L' "laţime="; l

după introducerea valorii corespunzătoare mesajului "lungime=" apărut la începutul unei linii, mesajul "lăţime=" îşi face apariţia la începutul liniei imediat următoare.

9.19. Instrucţiunea INPUT LINE

Instrucţiunea INPUT LINE oferă o modalitate de introducere de valori şir de caractere de la tastatură.

Formatul general este:

INPUT LINE listă; var. şir.

unde

- **„listă" este o constantă şir de caractere ce punctează acţiunea de introducere de date şi reprezintă un mesaj (un comentariu);**
- **„var. şir" este numele unei variabile şir de caractere ce va primi valoarea şir de caractere tastată.**

Argumentul „listă" nu este obligatoriu.

Atribuirea valorilor şir de caractere în urma unui INPUT LINE se consideră terminată în momentul tastării lui CR.

EXEMPLU:

Dacă la execuţia liniei program

10 INPUT LINE a$

se introduce de la tastatură linia de date

Carte

variabilei a$ i se atribuie valoarea constantă tip şir de caractere "Carte".

În contextul instrucţiunii INPUT LINE caracterul virgulă îşi pierde semnificaţia de separator şi este privit ca o constantă şir de caractere.

EXEMPLU:

Dacă la execuţia liniei program

10 INPUT a$

se introduce de la tastatură linia de date

desen, varianta 1

variabilei a$ i se atribuie valoarea „desen".

Dacă se înlocuieşte linia 10 anterior descrisă cu linia

10 INPUT LINE a$

prin introducerea aceleaşi linii de date, variabilei a$ i se atribuie valoarea şir de caractere "desen, varianta 1".

9.20. Instrucţiunea PLOT

Instrucţiunea PLOT desenează un punct pe ecran.

Formatul general este:

PLOT x, y

unde

— „x“ reprezintă distanţa faţă de extrema stîngă a ecranului şi se numeşte abscisa punctului;

— „y“ reprezintă distanţa faţă de baza ecranului şi se numeşte ordonata punctului.

Se ştie de la pct. 9.8 că partea utilizabilă a ecranului are 22 de linii şi 32 coloane.

22 linii × 32 coloane=704 poziţii de caracter.

Fiecare poziţie de caracter este la rîndul său un pătrat format din 8 ×8= 64 puncte. Punctele se numesc şi pixeli. Deci pe ecranul televizorului se vor putea desena (utiliza)

704×64=45056 puncte

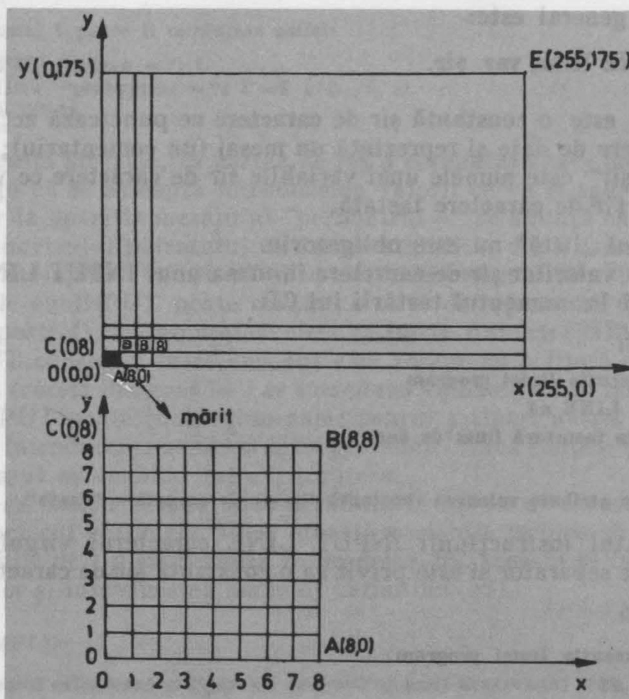


Fig. 9.23.

Poziţia unui pixel este specificată prin numerele x şi y.

Se mai spune că x şi y sînt coordonatele punctului respectiv. Matematic, coordonatele unui punct se descriu ca o pereche de numere între paranteze, astfel că pentru HC-85:

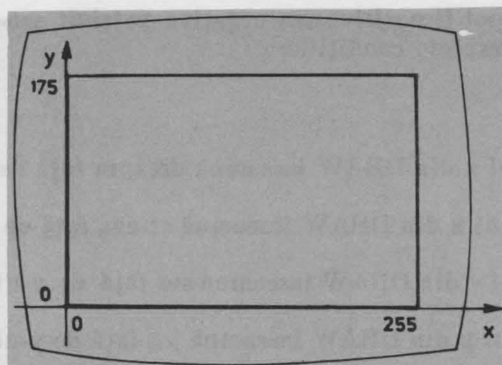


Fig. 9.24.

(0,0) — este punctul din extrema stînga jos a ecranului;

(255,0) — este punctul din extrema dreapta jos a ecranului;

(0,175) — este punctul din extrema stîngă sus a ecranului;

(255,175) — este punctul din extrema dreapta sus a ecranului.

Trebuie reţinut deci că, x şi y trebuie să respecte condiţiile:

$0 \leqslant x \leqslant 255$

$0 \leqslant y \leqslant 175$

Coordonatele x şi y din instrucţiunea PLOT pot fi numere sau expresii aritmetice a căror valoare trebuie să respecte condiţiile specificate înainte.

EXEMPLU:

Execuţia liniei program

10 PLOT 127,87

înseamnă desenarea unui singur punct în centrul ecranului.

9.21. Instrucţiunea DRAW

Instrucţiunea DRAW trasează de la ultimul punct desenat un segment de dreaptă sau o porţiune de cerc.

Formatul general este:

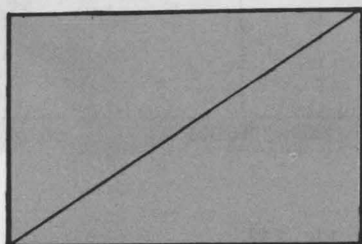
DRAW x, y, r

unde

— „x" reprezintă distanţa pe orizontală faţă de abscisa (x-ul) ultimului punct desenat;

— „y" reprezintă distanţa pe verticală faţă de ordonata (y-ul) ultimului punct desenat;

— „r" reprezintă numărul de radiani corespunzător lungimii arcului de cerc.


Fig. 9.25.

Instrucţiunea DRAW în care nu este specificat r trasează un segment de dreaptă.

EXEMPLU:

Programul următor trasează prima diagonală a ecranului (fig. 9.25).

```
10 PLOT 0,0
20 DRAW 255,175
```

Argumentele x şi y din DRAW pot fi pozitive sau negative potrivit sensului de trasare dorit şi trebuie să respecte condiţiile:

$-255 \leqslant x \leqslant 255$
$-175 \leqslant y \leqslant 175$

Sensul pozitiv pentru argumentul x din DRAW înseamnă dreapta faţă de x-ul ultimului punct desenat.

Sensul negativ pentru argumentul x din DRAW înseamnă stînga faţă de x-ul ultimului punct desenat.

Sensul pozitiv pentru argumentul y din DRAW înseamnă sus faţă de y-ul ultimului punct desenat.

Sensul negativ pentru argumentul y din DRAW înseamnă jos faţă de y-ul ultimului punct desenat.

EXEMPLE:

În programele următoare se trasează segmentul AB cu sensul de la A la B. Trebuie subliniat că în urma execuţiei, pe ecran este afişată numai linia continuă AB, celelalte fiind linii ajutătoare trasate în sprijinul înţelegerii modalităţii concrete de funcţionare a instrucţiunii exemplificate.

Programul 1 (fig. 9.26)

```
10 PLOT  50, 50
20 DRAW  100, 100
```

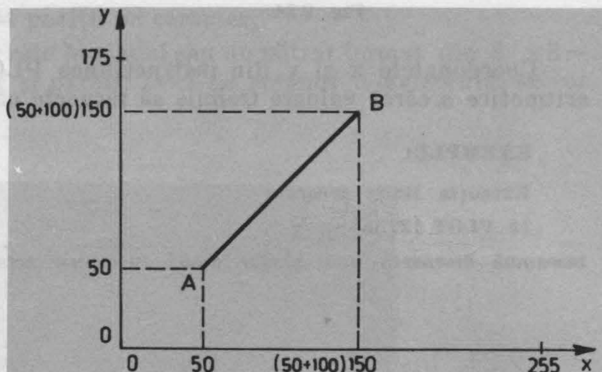


Fig. 9.26.

Programul 2 (fig. 9.27)

```
10 PLOT  100, 100
20 DRAW  50, 50
```

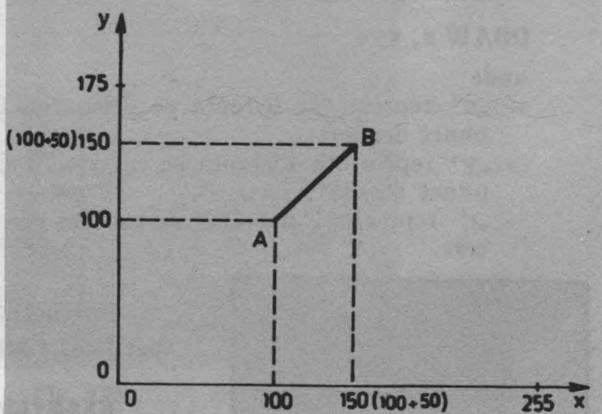


Fig. 9.27.

Programul 3

```
10 PLOT  200, 160
20 DRAW  —130, —70
```

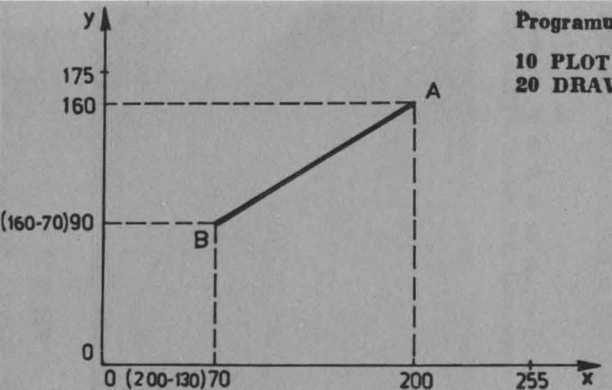


Fig. 9.28.

Programul 4

```
10 PLOT   180, 20
20 DRAW   —90, 100
```

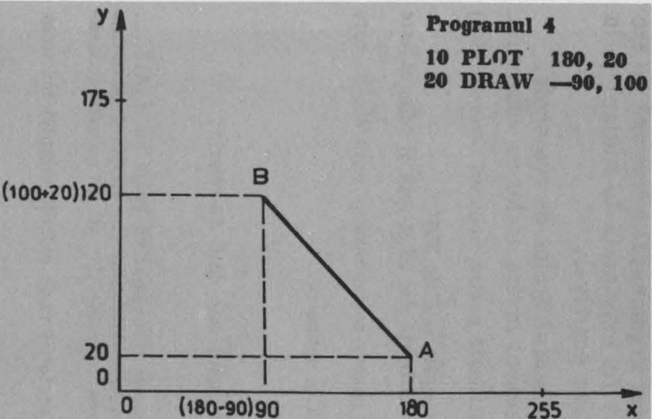


Fig. 9.29.

Programul 5

```
10 PLOT  30, 60
20 DRAW  120, —40
```

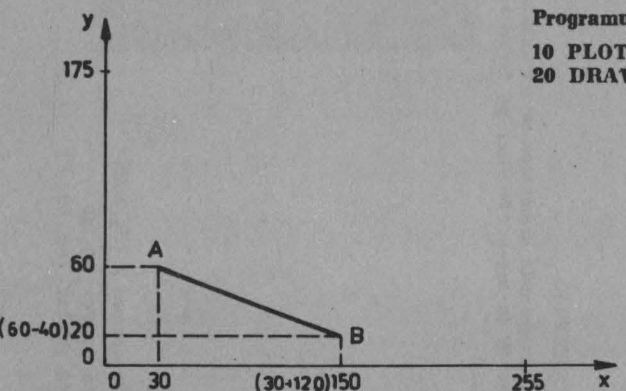


Fig. 9.30.

Programul 6

```
10 PLOT (xa, ya)
20 DRAW  (x, y)
```

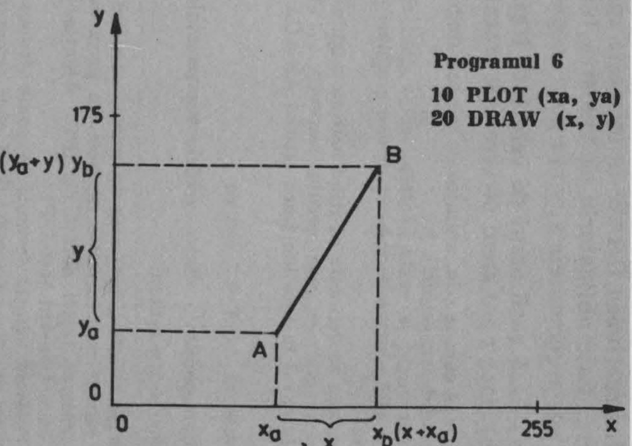


Fig. 9.31.

Instrucţiunea DRAW în care este specificat argumentul r trasează un arc de cerc. Este obligatoriu ca valoarea lui r să fie exprimată în radiani şi în acest caz argumentele x, y şi r pot fi negative sau pozitive:

a) dacă $r<0$ arcul de cerc este trasat în sensul acelor de ceasornic;

b) dacă $r>0$ arcul de cerc este trasat în sensul invers acelor de ceasornic.

După cum se ştie, unitatea de măsură frecvent utilizată pentru unghiuri (arce de cerc) este gradul sexagesimal.

Un unghi la centru corespunzător unui cerc complet este de 360°.

O altă unitate de măsură pentru unghiuri (arce) utilizată des în ştiinţă şi tehnică este radianul.

Acesta se defineşte ca fiind măsura unghiului la centru corespunzător unui arc de cerc egal ca lungime cu raza cercului respectiv.

În cazul în care lungimea arcului AB=OA=raza se spune că:

măsura $\widehat{AOB} = 1$ radian

Corespondenţa dintre gradele sexazecimale şi radiani este dată de relaţia:

$360° = 2\pi$ radiani

unde π (se citeşte „ pi") este un număr care are valoarea aproximativă egală cu 3,1416 .... Acest număr este introdus în memoria permanentă a calculatorului (în ROM) şi se obţine prin apăsarea tastei notată cu „PI". Atenţie, tastarea literelor „P" şi „I" nu este echivalentă cu acţionarea tastei PI!

Folosind această formulă se poate deduce (cu ajutorul unor calcule simple) mărimea în radiani a oricărui unghi, după cum urmează:

a) $180° = \frac{360°}{2} = \frac{2\pi}{2}$ radiani $= \pi$ radiani

b) $60° = \frac{360°}{6} = \frac{2\pi}{6}$ radiani $= \frac{\pi}{3}$ radiani $= 0.333$ radiani

c) $70° = 60° + 10° = \frac{360°}{6} + \frac{360°}{36} = \left(\frac{\pi}{3} + \frac{2\pi}{36}\right)$ radiani $= \left(\frac{\pi}{3} + \frac{\pi}{18}\right)$ radiani $=$

$=(0.333+0.055)\ \pi$ radiani $=0.388\ \pi$ radiani

EXEMPLE:

În programele următoare se trasează semicercul AB cu sensul de la A la B. Trebuie subliniat că în urma execuţiei pe ecran apare numai semicercul AB, celelalte fiind linii ajutătoare.

Programul 7 (fig.9.32)

```
10 PLOT  20, 30
20 DRAW  100, 60, PI
```

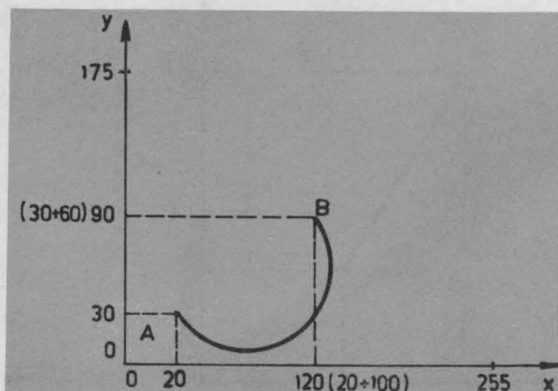


Fig. 9.32.

Programul 8

```
10 PLOT  90, 70
20 DRAW  50, 20, PI
```

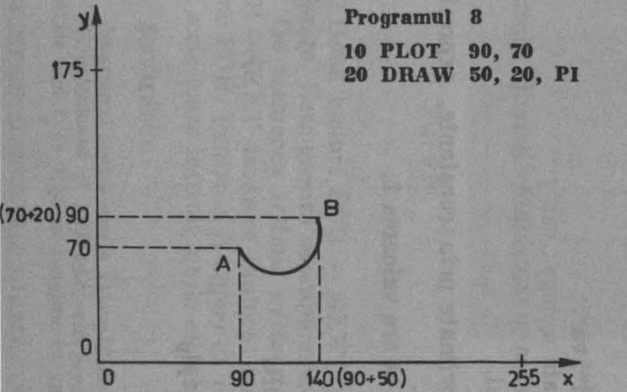


Fig. 9.33.

Programul 9

```
10 PLOT  150, 60
20 DRAW  —100, —30, PI
```

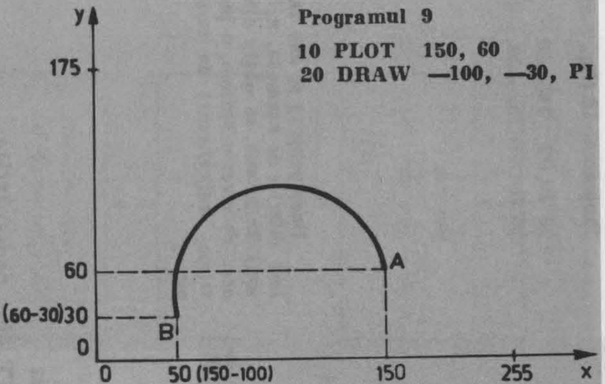


Fig. 9.34.

Programul 10

```
10 PLOT  50, 50
20 DRAW  —20, 20, PI
```

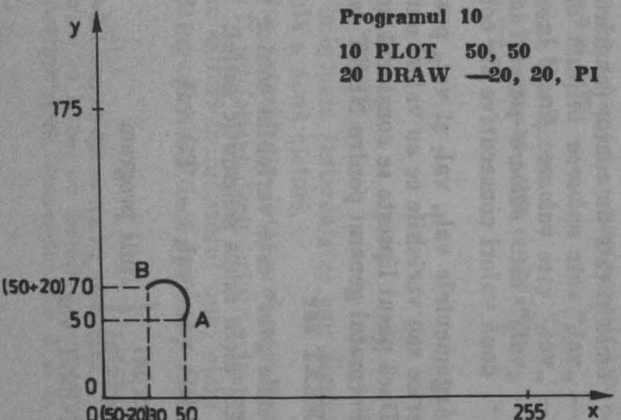


Fig. 9.35.

Programul 11

```
10 PLOT  110, 80
20 DRAW  70, —10, PI
```

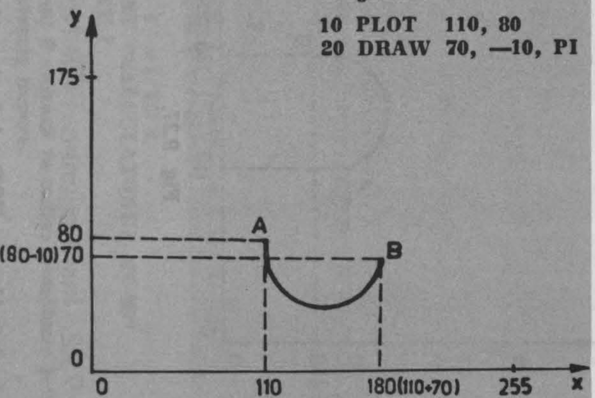


Fig. 9.36.

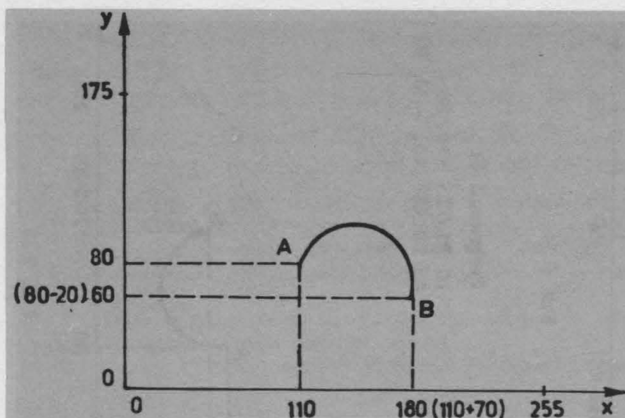


Fig. 9.37.

Programul 12 (fig. 9.37)

```
10 PLOT   110, 80
20 DRAW 70, —20, —PI
```

Dacă pentru r se dau valori diferite de exemplu, π/4, π/2, 3π/2, etc., se obţin alte arce de cerc: o optime, o pătrime, trei sferturi de cerc, etc.

9.22. Instrucţiunea FOR-NEXT

Instrucţiunea FOR şi instrucţiunea NEXT sînt nedespărţite.

Prima defineşte începutul unui ciclu, iar a doua sfîrşitul lui.

Prin ciclu se înţelege execuţia de un număr prestabilit de ori a grupului de instrucţiuni cuprinse între FOR şi NEXT.

Formatul general pentru FOR este:

FOR var=val_i TO val_f STEP val_p

unde

- „var". este o variabilă numerică simplă (numele său trebuie să fie obligatoriu format dintr-o singură literă) ce reprezintă variabila de ciclu şi este un contor (numărător);
- „val_i" este *valoarea iniţială* dată lui „var";
- „val_f" este *valoarea finală* la care poate ajunge „var";
- „val_p" este *valoare pasului* sau valoarea ce reprezintă diferenţa dintre două valori consecutive ale contorului.

Argumentele val_i, val_f şi val_p pot fi exprimate prin constante, expresii numerice sau variabile ce au valori numerice.

Dacă pasul lipseşte se consideră implicit că are valoarea 1.

Formatul general pentru NEXT este:

NEXT var

unde „var" este variabila contor din FOR.

Un ciclu arată schematic astfel:

FOR var=val_i TO val_f STEP val_p

corpul ciclului { linii program

NEXT var

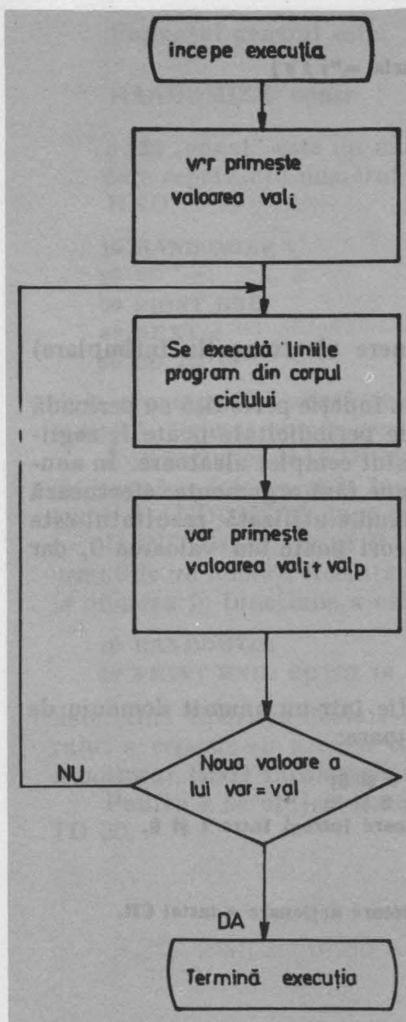


Fig. 9.38.

Execuţia unui ciclu constă din etapele descrise în fig. 9.38.

EXEMPLE:

Programul 1 afişează de 5 ori textul "CALCULATORUL HC-85"

```
10 FOR I =1 TO 5
20 PRINT "CALCULATORUL HC-85"
30 NEXT I
```

Programul 2 afişează în ordine crescătoare primele 10 numere naturale impare.

```
10 FOR i = 1 TO 19 STEP 2
20 PRINT i
30 NEXT i
```

Programul 3 afişează în ordine descrescătoare primele 10 numere naturale.

```
10 FOR n = 10 TO 1 STEP — 1
20 PRINT n
30 NEXT n
```

Programul 4 trasează graficul funcţiei SIN pentru valori între 0 şi 2 PI.

```
10 FOR n = 0 TO 255
20 PLOT n, 88 +80 * SIN (n/128 * PI)
30 NEXT n
```

Două cicluri FOR – NEXT pot fi imbricate (intercalate) sau complet separate. Este greşită suprapunerea parţială a două cicluri.

De asemenea trebuie evitat saltul din exterior în interiorul unei bucle FOR–NEXT deoarece contorul nu poate fi iniţializat decît printr-o instrucţiune FOR. Pentru a fi siguri că nu se fac salturi în interiorul unui ciclu se pot scrie toate instrucţiunile ciclului pe o singură linie (dacă spaţiul permite).

EXEMPLU:

Programul acesta este o exemplificare de două cicluri FOR — NEXT imbricate. El calculează şi tipăreşte aria dreptunghiurilor cu lăţime ce variază între 1 şi 100, crescător din 1 în 1 şi cu lungime ce variază între 200 şi 20, descrescător din 2 în 2.

```
10 FOR i        = 1 TO 100 STEP 1
20 FOR j        = 200 TO 2 STEP — 2
30 PRINT "laţimea = "; i; "lungimea = "; j; "aria ="; i * j
40 PRINT
50 NEXT j
60 NEXT i
```

9.23. Instrucţiunea RND

Prin instrucţiunea RND se generează numere aleatoare (la întîmplare) fără a folosi tastatura.

RND nu este o funcţie complet aleatoare ci o funcţie periodică cu perioadă suficient de mare (65535), astfel încît efectul de periodicitate poate fi neglijat. În cadrul unei perioade, numerele generate sînt complet aleatoare. În anumite privinţe, RND se comportă ca o instrucţiune fără argumente: efectuează calcule şi produce un rezultat. De fiecare dată cînd e utilizată, rezultatul este un număr aleator nou, cuprins între 0 şi 1 (uneori poate lua valoarea 0, dar niciodată 1).

Formatul general este:

RND

Dacă se doreşte ca numerele aleatoare să fie într-un anumit domeniu de valori se poate proceda ca în exemplele următoare:

5 * RND	**generează numere între 0 şi 5;**
1.3+0.7 * RND	**produce numere între 1. 3 şi 2;**
1+INT (RND * 6)	**furnizează numere aleatoare întregi între 1 şi 6.**

EXEMPLE:

Programul 1 simulează aruncarea zarurilor după fiecare acţionare a tastei CR.

```
10 CLS
20 FOR n = 1 TO 2
30 PRINT 1 +INT (RND * 6); "  ";
40 NEXT n
50 INPUT a$
60 GO TO 20
```

Programul 2 trasează aleator un punct pe ecran după fiecare acţionare a tastei CR.

```
10 PLOT INT (RND * 255), INT (RND * 175)
20 INPUT a$
30 GO TO 10
```

9.24. Instrucţiunea RANDOMIZE

Instrucţiunea RANDOMIZE este utilizată împreună cu RND şi face ca RND să pornească dintr-un punct anumit al secvenţei de generare a numerelor.

Formatul general este:

RANDOMIZE const

unde „const" este un număr cuprins între 0 şi 65535 (perioada lui RND) care reprezintă numărul de ordine al viitorului apel al unei instrucţiuni RND.

```
10 RANDOMIZE 1
20 FO[illegible]
30 PRINT RND
40 NEXT n
50 GO TO 10
```

Linia program 10 fixează pentru RND condiţia de a începe de la numărul de ordine 1. Primul număr generat de RND este întotdeauna 0.0022735596.

Instrucţiunea RANDOMIZE poate fi plasată oriunde în program, dar este indicat să se folosească la începutul lui. Practica recomandă utilizarea instrucţiunii RANDOMIZE după punerea la punct a programului.

RANDOMIZE, ca şi RANDOMIZE 0, are efect diferit de RANDOMIZE urmat de un număr. Această formă a instrucţiunii se referă la timpul trecut de la punerea în funcţiune a calculatorului.

```
10 RANDOMIZE
20 PRINT RND: GO TO 10
```

determină tipărirea aceluiaşi număr. Deoarece timpul de lucru al calculatorului a crescut cu aceeaşi cantitate la fiecare execuţie a lui RANDOMIZE, următorul RND furnizează aproximativ acelaşi rezultat.

Pentru a se obţine o secvenţă aleatoare se înlocuieşte GO TO 10 cu GO TO 20.

9.25. Instrucţiunea CIRCLE

Instrucţiunea CIRCLE desenează pe ecran conturul unui cerc.

Formatul general este:

CIRCLE x, y, r

unde

— „x" şi „y" sînt coordonatele centrului cercului şi au aceeaşi semnificaţie ca la pct. 9.20;

— „r" este lungimea în puncte (pixeli) a razei cercului.

Argumentul r din CIRCLE trebuie să fie pozitiv (dacă semnul este minus el este ignorat), iar coordonatele x şi y pot fi constante, expresii numerice sau variabile cu valori numerice, ce trebuie să respecte condiţiile:

$0 \leqslant x \pm r \leqslant 255$
$0 \leqslant y \pm r \leqslant 175$

EXEMPLE:

Trebuie reţinut că după execuţia următoarelor programe concepute în scopul exemplificării instrucţiunii grafice CIRCLE, pe ecran apare numai imaginea desenată cu negru, axele de coordonate şi restul liniilor ajutătoare fiind trasate numai pentru uşurarea înţelegerii.

Programul 1 deseneaza conturul unui cerc, centrul său şi una din tangentele sale (fig. 9.39)

```
10 CIRCLE 100, 100, 50
20 PLOT   100, 100
30 PLOT   0,150
40 DRAW   200, 0
```

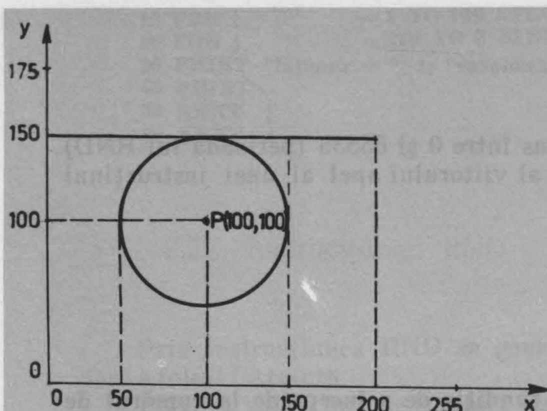


Fig. 9.39.

Prin linia 10 se trasează cercul cu centrul în punctul de coordonate (100, 100) şi de rază 50.

Prin linia 20 se trasează punctul ce reprezintă centrul cercului.

Prin liniile 30 şi 40 se trasează segmentul de dreaptă AB, tangent la cercul deja realizat.

Programul 2 desenează un disc negru în mijlocul ecranului (de fapt 50 de cercuri concentrice) (fig. 9.40)

```
10 FOR i = 1 TO 50
20 CIRCLE 127, 87, i
30 NEXT i
```

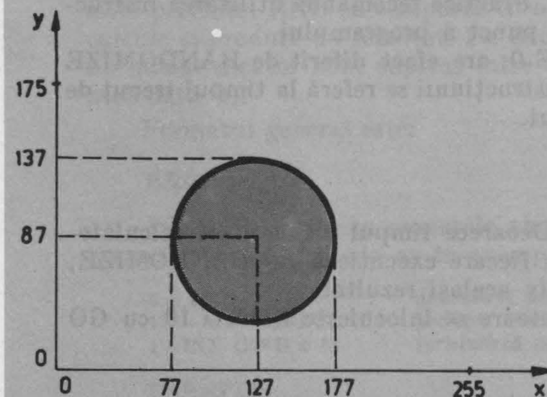


Fig. 9.40.

Programul 3 desenează 5 cercuri concentrice (primul cerc este numai un punct ce reprezintă centrul cercurilor) (fig. 9.41)

```
10 FOR  i = 0 TO  50  STEP  10
20 CIRCLE 127, 87, i
30 NEXT i
```

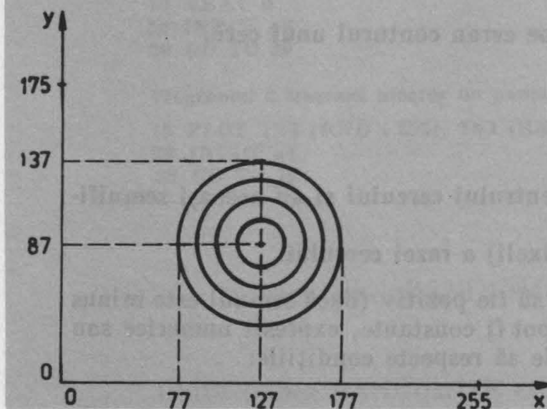


Fig. 9.41.

9.26. Instrucţiunea OVER

Diferite figuri cu diverse forme se pot trasa folosind cu ingeniozitate instrucţiunile grafice PLOT, DRAW sau CIRCLE. În procesul proiectării apar situaţii care impun modificarea unei anumite părţi dintr-o figură, Pentru asemenea cazuri a fost concepută instrucţiunea OVER.

Instrucţiunea OVER oferă posibilitatea ştergerii unui punct, unei

drepte, unui arc de cerc sau unui cerc, după cum este folosită alături de PLOT, DRAW sau CIRCLE.

Formatul general este:

OVER 1

Avantajul instrucţiunii OVER constă în faptul că permite ştergerea unei părţi dintr-o figură fără a afecta restul figurilor de pe ecran.

Atît instrucţiunea OVER cît şi instrucţiunea CLS efectuează o ştergere, fiecare însă în alt mod, prima afectează o parte din desen, a doua tot ecranul.

Trebuie ţinut seama de constatarea că o dreaptă sau un arc de cerc se şterg complet în aceeaşi direcţie şi sens în care au fost trasate.

EXEMPLE:

Deşi desenul realizat pentru fiecare program constă în mai multe puncte, drepte etc., după execuţie, numai imaginea neagră continuă rămîne pe ecran (fără axele de coordonate şi celelalte linii ajutătoare).

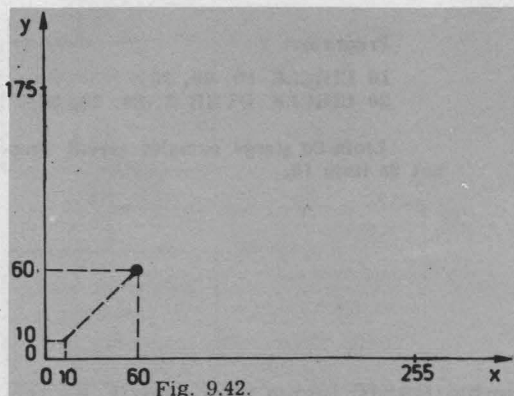


Fig. 9.42.

Programul 1 (fig. 9.42)

```
10 PLOT 10, 10
20 DRAW 50, 50
30 DRAW OVER 1, —50, —50
```

Linia 30 şterge dreapta trasată de liniile program 10 şi 20 cu excepţia punctului de coordonate (60, 60).

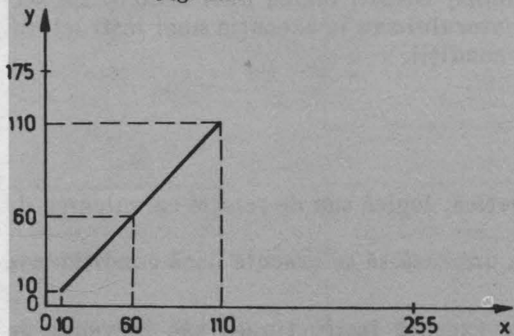


Fig. 9.43.

Programul 2 (fig. 9.43)

```
10 PLOT 10, 10
20 DRAW 50, 50
30 DRAW OVER 1, 50, 50
```

Prin linia program 30 nu se şterge dreapta de pe ecran, ci se prelungeşte, pentru că nu s-a respectat convenţia referitoare la sensul de ştergere (să fie identic cu sensul de trasare a dreptei).

Programul 3

Dreapta trasată de liniile program 10 şi 20 din programul 2 se şterge prin adăugarea liniilor.

```
30 PLOT OVER 1, 10, 10
40 DRAW OVER 1, 50, 50
```

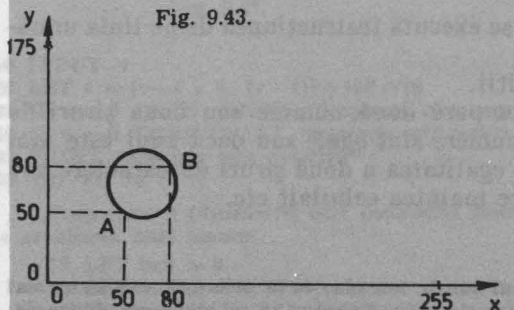


Fig. 9.44.

Programul 4 (fig. 9.44)

```
10 PLOT 50, 50
20 DRAW 30, 30, PI
30 DRAW OVER 1,—30, —30, PI
```

Linia 30 nu şterge semicercul desenat de liniile 10 şi 20, ci completează desenul cu cealaltă parte a semicercului.

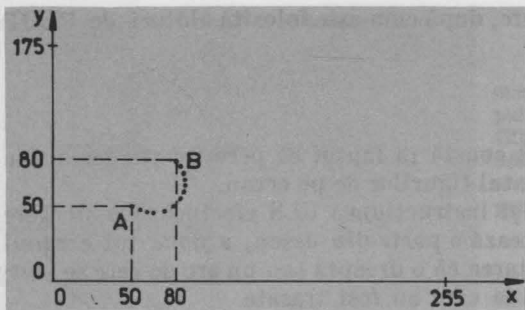


Fig. 9.45.

Programul 5 (fig. 9.45)

```
10 PLOT   50, 50
20 DRAW 30, 30, PI
30 DRAW OVER 1, —30, —30, PI
```

Linia 30 şterge semicercul trasat cu ajutorul liniilor 10 şi 20, dar lasă pe ecran urma cîtorva puncte (în număr de 13) semn că schimbîndu-se sensul, nu se parcurge chiar acelaşi traseu.

Programul 6

```
10 PLOT   50, 50
20 DRAW   30, 30, PI
30 PLOT   OVER 1, 50, 50
40 DRAW   OVER 1, 30, 30, PI
```

Liniile 30 şi 40 şterg complet semicercul trasat de liniile program 10 şi 20.

Programul 7

```
10 CIRCLE 80, 50, 30
20 CIRCLE OVER 1, 80, 50, 30
```

Linia 20 şterge complet cercul desenat de linia 10.

9.27. Instrucţiunea IF

Instrucţiunea IF transpune în limbaj BASIC luarea unei decizii. Ea serveşte la realizarea unui transfer al contorului sau la execuţia unei instrucţiuni în funcţie de valoarea logică a unei condiţii.

Formatul general este:

IF cond THEN instr

unde

— „cond" este o expresie aritmetică, logică sau de relaţie cu valoarea de adevărat sau fals;

— „instr" este instrucţiunea ce urmează să se execute dacă condiţia este adevărată.

Dacă condiţia este adevărată se execută instrucţiunea sau secvenţa de instrucţiuni scrisă după THEN.

Dacă condiţia nu este adevărată se execută instrucţiunea de pe linia următoare a lui IF.

Se pot imagina tot felul de condiţii.

Una dintre cele mai simple, compară două numere sau două şiruri de caractere. Se poate testa dacă două numere sînt egale sau dacă unul este mai mare decît celălalt. Se poate testa şi egalitatea a două şiruri de caractere sau dacă în ordinea alfabetică, unul apare înaintea celuilalt etc.

EXEMPLE:

Programul 1 compară mereu dacă primul număr introdus de la tastatură este strict mai mare decît al doilea număr introdus tot de la tastatură şi tipăreşte pe cel mai mare dintre ele. (fig. 9.46).

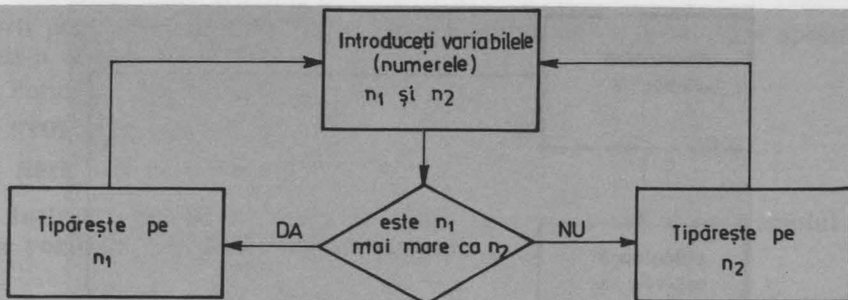


Fig. 9.46.

```
10 INPUT "nr. 1", a
20 INPUT "nr. 2", b
30 IF a > b THEN PRINT
"nr. 1 este mai mare", a: GO TO 10
40 PRINT "nr. 2 este mai
mare", b
50 GO TO 10
```

Programul 2 tipăreşte primele 20 numere naturale. (fig. 9.47)

```
10 LET a =0
20 LET a = a +1
30 IF  a =20 THEN GO TO 60
40 PRINT a
50 GO TO 20
60 PRINT a
70 PRINT "SFIRSIT"
```

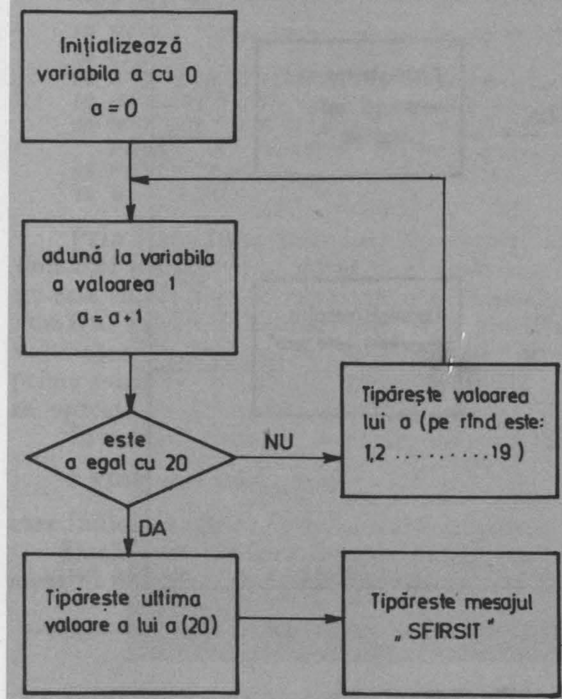


Fig. 9.47.

Programul 3 afişează dacă rezultatul unei expresii aritmetice este negativ, zero sau pozitiv. (fig. 9.48)

În schema logică aleasă se observă că un al treilea bloc de decizie (în care s-ar compara dacă „e" este mai mare ca 0) dispare, deoarece dacă „e" nu este mai mic ca zero şi nici egal cu zero atunci sigur este mai mare ca zero.

```
10 INPUT v
20 LET e = (v—v * 5/(v—1))+150*v↑2
30 IF e < 0 THEN PRINT "expresia este negativa": GO TO 10
40 IF e = 0 THEN PRINT "expresia este zero": GO TO 10
50 PRINT "expresia este pozitiva"
60 GO TO 10
```

Programul 4 (distractiv) care determină frecvenţa de apariţie a banului sau a stemei la aruncarea unei monezi.

```
10 LET ban = 0
20 LET stema = 0
30 LET moneda = INT (RN D*2)
40 IF  moneda =0 THEN LET ban = ban + 1
```

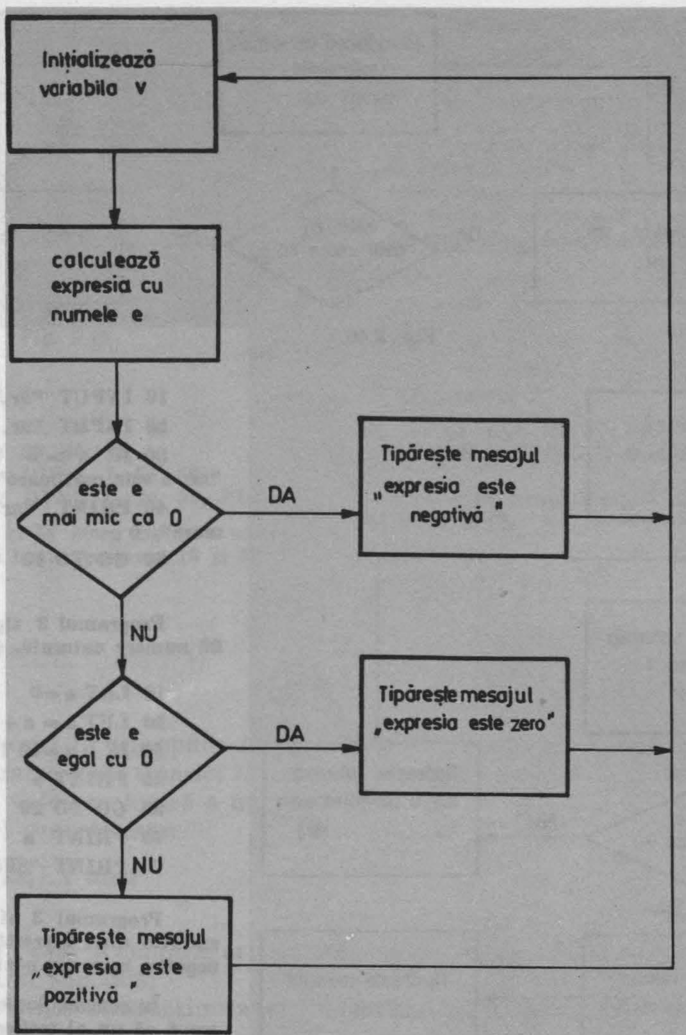


Fig. 9.48.

```
50 IF moneda =1 THEN LET stema = stema +1
60 PRINT ban; ", "; stema
70 IF stema > 0 THEN PRINT ban / stema;
80 PRINT
90 GO TO 20
```

Dacă timpul de rulare a programului este suficient de mare, raportul ban / stema devine aproximativ 1, deoarece numerele aleatoare generate cu RND sînt uniform repartizate în intervalul (0, 1).

9.28. Instrucţiunea STOP

Instrucţiunea STOP întrerupe temporar execuţia unui program. Execuţia poate fi reluată începînd cu linia program imediat următoare

opririi prin comanda CONTINUE sau începînd cu linia program specificată printr-o comandă GO TO.

Formatul general este:

STOP

(fără nici un argument).

Instrucţiunea STOP poate să apară în orice punct al programului şi pe orice poziţie a unei linii program multiple.

Exemplu :

Programul următor verifică dacă se cunosc numerele prime din intervalul [20, 50] şi totodată afişează aproximativ în mijlocul ecranului numărul prim introdus.

```
10 INPUT "alege un nr. prim între 20 şi 50"; N
20 CLS
30 IF N < 20 THEN PRINT "nr. ales este mai mic ca 20"; N: GO TO 10
40 IF N=51 THEN PRINT "nr. ales este mai mare ca 50"; N: GO TO 10
50 IF N=23 OR N=29 OR N=31 OR N=37 OR N=41 OR N=43 OR N=47 THEN
   PRINT AT 11, 16; N : STOP
60 PRINT "nr. ales nu este prim"
70 GO TO 10
```

Prin linia 10 se introduce un număr, iar linia 20 şterge ecranul (de fapt numărul introdus); numărul va fi tipărit imediat după mesajul de eroare dacă nu este corect ales, în caz contrar în mijlocul ecranului. Linia 30 verifică dacă numărul introdus este mai mic, iar linia 40 dacă este mai mare faţă de intervalul ales. În program, linia 50 compară numărul introdus cu toate numerele prime posibile şi dacă alegerea este corectă se afişează acel număr, iar execuţia se opreşte la întîlnirea instrucţiunii STOP.

În această situaţie, în partea de jos a ecranului apare mesajul

9 STOP startement, 50: 3

care indică că oprirea este cauzată de a treia instrucţiune din linia 50.

Dacă numărul ales nu este corect, se execută şi linia 60 care afişează mesajul de eroare, iar prin linia 70 se reia programul de la început.

9.29. Instrucţiunea CONTINUE

Instrucţiunea CONTINUE se foloseşte împreună cu instrucţiunea STOP pentru a se relua o execuţie.

CONTINUE este folosită sub formă de comandă.

Formatul general este:

CONTINUE

(fără nici un argument).

Este util de reţinut că STOP şi CONTINUE împreună, oferă un mijloc eficient de punere la punct a programelor (în special cele de dimensiuni mari

sau de complexitate deosebită) prin testarea eşalonată a liniilor sau a unui grup de linii.

Ca exemplu, în cazul programului din capitolul anterior, după introducerea unui număr prim corect şi afişarea lui, se poate relua execuţia (fără a tasta RUN) folosindu-se comanda CONTINUE.

9.30. Instrucţiunea DIM

La pct. 8.4.3. s-a definit noţiunea de variabilă indexată. Unei variabile indexate îi corespunde un număr de şiruri de caractere egal cu numărul indicilor săi. În locul denumirii de variabilă indexată este folosită denumirea de tablou (sau matrice), iar numărul indicilor corespunde numărului dimensiunilor tabloului.

O variabilă indexată poate fi numerică sau alfanumerică drept pentru care elementele tabloului pot fi constante de orice tip.

Un tablou numeric este reprezentat printr-un nume format dintr-o singură literă, pe cînd un tablou alfanumeric este reprezentat tot printr-un nume format dintr-o singură literă urmată de caracterul $.

Pentru a putea fi utilizat, unui tablou trebuie să i se rezerve un spaţiu în memorie înainte de completarea sa.

Spaţiul necesar unui tablou este echivalent cu spaţiul total necesar memorării tuturor elementelor sale. Numărul elementelor unui tablou este dat de rezultatul produsului dintre valorile dimensiunilor sale.

Instrucţiunea DIM serveşte pentru declararea dimensiunilor unui tablou.

Formatul general este:

DIM var (dim1, dim2, ...)

unde

— „var" este numele tabloului;

— „dim1, dim2, ..." sînt dimensiunile tabloului şi sînt exprimate prin constante, expresii aritmetice sau variabile ce au ca valoare un număr natural diferit de 0.

Printr-o instrucţiune DIM se definesc numai dimensiunile unui singur tip de tablou.

Ea are următorul efect:

— rezervă spaţiu pentru fiecare element al tabloului (adică const1 × const 2 × ... = numărul de locuri);

— iniţializează toate elementele tabloului cu 0;

— şterge orice tablou cu acelaşi nume.

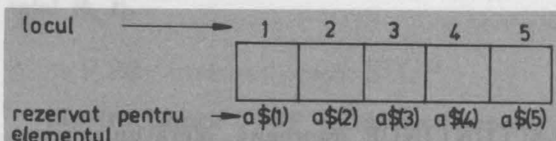


Fig. 9.49.

EXEMPLE:

Linia program
10 DIM a$ (5)

rezervă 5 locuri în memorie pentru elementele tabloului a$ (fig. 9.49).

Linia program

20 DIM P(3, 2)

rezervă 6 locuri (6 de la 3 ×2) în memorie pentru elementele tabloului P.

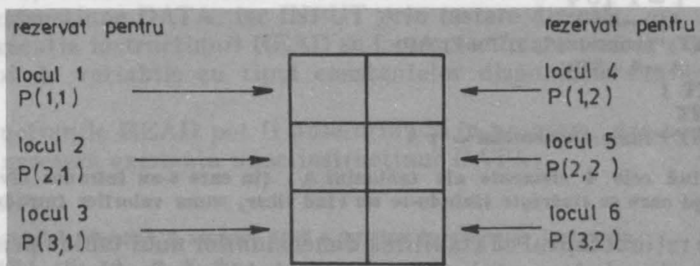


Fig. 9.50.

Pentru tabloul cu numele a din exemplul 1 (prevăzut cu un singur indice) s-a rezervat un şir de 5 locuri consecutive, deoarece este definit ca o succesiune obişnuită de 5 elemente.

Pentru tabloul cu numele P din exemplul 2 (prevăzut cu 2 indici) se rezervă în final tot un şir de locuri, procedîndu-se astfel: în prima fază se face rezervarea a 3 locuri pentru elementele din prima coloană, după care imediat în continuarea acestora, alte 3 locuri pentru elementele din cea de-a doua coloană. Utilizînd acest model se ajunge la rezervarea celor 6 locuri consecutive necesare introducerii tuturor elementelor tabloului P.

Procedeul descris rămîne valabil în cazul oricărui tablou cu un număr oarecare de indici.

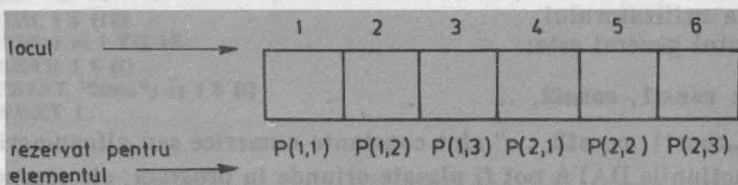


Fig. 9.51.

Linia program

30 DIM r (2, 3, 2)

rezervă 12 locuri (12 de la 2 ×3 ×2) în memorie pentru elementele tabloului r.

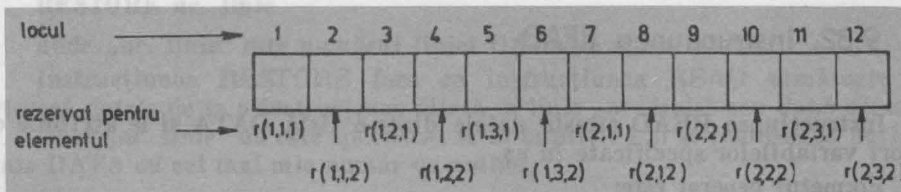


Fig. 9.52.

Valorile dimensiunilor se pot exprima fie printr-o constantă, fie printr-o variabilă astfel că efectul liniilor 20 şi 30 din exemplele anterioare, este identic cu cel din următoarea secvenţă de instrucţiuni:

```
5 LET  a = 2
10 LET  b = 3
20 DIM  P(b, a)
30 DIM  r(a, b, a)
```

Programul

```
10 DIM A(4)
20 LET A=0
30 FOR i = 1 TO 4
40 LET A(i)=5 ↑ i
50 PRINT "elementul"; i; "="; A(i)
60 LET A=A+A(i)
70 NEXT i
80 PRINT
90 PRINT "suma elementelor ="; A
```

afişează pe rînd cele 4 elemente ale tabloului A (în care s-au introdus diverse puteri ale lui 5), după care se tipăreşte lăsîndu-se un rînd liber, suma valorilor introduse.

Trebuie reţinut faptul că stabilirea dimensiunilor unui tablou prin instrucţiunea DIM se face la începutul unui program şi obligatoriu înaintea prelucrării elementelor acelui tablou.

9.31. Instrucţiunea DATA

Instrucţiunea DATA introduce date în timpul execuţiei programului fără intervenţia utilizatorului.

Formatul general este:

DATA const1, const2, ...

unde „const1, const2, ..." sînt constante numerice sau alfanumerice.

Instrucţiunile DATA pot fi plasate oriunde în program, ele comportîndu-se ca o listă unică realizată prin concatenarea tuturor instrucţiunilor DATA din program (lista DATA). Practica recomandă gruparea instrucţiunilor DATA la sfîrşitul programului pentru verificare rapidă în momentul testării sau execuţiei.

Instrucţiunea DATA se foloseşte împreună cu instrucţiunea READ.

9.32. Instrucţiunea READ

Instrucţiunea READ citeşte datele dintr-o listă DATA şi le atribuie ca valori variabilelor specificate în ea.

Formatul general este:

READ var1, var2, ...

unde „var1, var2, ..." sînt variabile simple sau indexate de tip numeric sau şir de caractere.

Cînd calculatorul citeşte prima variabilă cu READ, ei îi este asociată prima valoare din lista DATA, şi aşa mai departe.

Dacă se încearcă citirea mai multor variabile decît numărul valorilor din lista DATA, atunci apare eroare.

Instrucţiunile READ şi INPUT au acelaşi efect, respectiv acela de a introduce date pentru execuţia programului.

Diferenţa dintre ele rezidă din modul de funcţionare: READ ia datele dintr-o instrucţiune DATA, iar INPUT prin tastare directă.

La execuţia instrucţiunii READ se face o verificare strictă a corespondenţei tipului de variabile cu tipul constantelor disponibile din instrucţiunea DATA.

Instrucţiunile READ pot fi puse oriunde în program, dar pentru ele este neapărat necesară existenţa unor instrucţiuni DATA.

EXEMPLE:

Programul 1 afişează 5 numere fără a utiliza în execuţie tastatura.

```
10 DATA  5, 10,—3, 7, 2
20 READ  a, b, c, d, e
30 PRINT "nr. 1 = "; a
40 PRINT "nr. 2 = "; b
50 PRINT "nr. 3 = "; c
60 PRINT "nr. 4 = "; d
70 PRINT "nr. 5 = "; e
```

Programul 2 afişează lunile anului.

```
10 DATA "ian", "febr", "mart", "mai", "iunie", "iulie", "aug", "sept". "oct",
"nov", "dec"
20 DIM l $ (12)
30 FOR i = 1 TO 12
40 READ l $ (i)
50 PRINT "luna"; i; l $ (i)
60 NEXT i
```

9.33. Instrucţiunea RESTORE

Instrucţiunea RESTORE permite refolosirea prin READ a datelor introduse prin DATA.

Formatul general este:

RESTORE nr. linie

unde „nr. linie“ este numărul liniei DATA ce se va refolosi.

Instrucţiunea RESTORE face ca instrucţiunea READ următoare să citească datele de la o instrucţiune aflată la linia „nr. linie“ sau după aceasta.

Dacă „nr. linie“ nu este specificat se ia implicit 1 şi se refoloseşte de fapt linia DATA cu cel mai mic număr de ordine.

EXEMPLU:

```
10 READ a, b
20 PRINT a, b
30 RESTORE 10
40 READ x, y, z
50 PRINT x, y, z
60 DATA 1, 2, 3
70 STOP
```

Pentru că în linia 30 este specificată o instrucţiune RESTORE, citirea variabilelor la liniile 10 şi 40 utilizează aceeaşi linie de date (linia cu numărul de ordine 60) astfel că variabilele primesc valorile:

```
a =1        b =2
x =1        y =2        c =3
```

9.34. Instrucţiunea DEF FN

Spre deosebire de funcţiile standard (funcţii matematice şi funcţii pe şir de caractere), limbajul BASIC admite şi funcţii definite de utilizator. Ele sînt cunoscute sub numele de funcţii utilizator.

Funcţiile utilizator au sens numai pe parcursul folosirii programului în care au fost definite.

Instrucţiunea DEF FN face posibilă definirea unei funcţii utilizator.

Formatul general este:

DEF FN nume (var1, var2, ...) =expresie

unde

— „nume" este numele dat funcţiei;

— „var1, var2 ..." reprezintă parametri formali ai funcţiei şi sînt variabile simple sau indexate;

— „expr" este expresia ce dă corespondenţa dintre elementele funcţiei.

Funcţiile utilizator sînt de două feluri potrivit datelor pe care le manevrează:

— numerice
sau
— alfanumerice.

Tipul funcţiilor utilizator rezultă din argumentul „nume" din DEF FN şi anume:

— o singură literă pentru funcţiile numerice şi

— o literă urmată de caracterul $ pentru funcţiile alfanumerice.

Forma sub care este apelată în program funcţia utilizator este:

FN nume (arg1, arg2 ...)

EXEMPLE:

Definirea funcţiei de ridicare la pătrat se face astfel:

DEF FN f(x) =x * x

Afişarea valorii acestei funcţii în punctul 5 generează următoarea secvenţă de linii program:

```
10 DEF FN f(x) = x * x
20 LET x=5
30 LET t=FN f(x)
40 PRINT t
```

Programul

```
10 DEF FN t(i) =val +i/2
20 LET val = 0
30 PRINT FN t(30)
40 LET val = 10
50 PRINT FN t(30)
```

afişează pe rînd numerele 15 şi respectiv 25 ce corespund valorilor funcţiei în următoarele condiţii:

$$\left.\begin{array}{l} val = 0 \\ i = 30 \end{array}\right\} \Rightarrow t(30) = 0 + 30/2 = 0 + 15 = 15$$

$$\left.\begin{array}{l} val = 10 \\ i = 30 \end{array}\right\} \Rightarrow t(30) = 10 + 30/2 = 10 + 15 = 25$$

9.35. Instrucţiunea GOSUB–RETURN

Practica a generat necesitatea repetării de un număr de ori a unui grup de linii program în cadrul execuţiei aceluiaşi program.

Acest grup de linii formează un subprogram sau o subrutină.

Manipularea unei subrutine se face cu ajutorul a două instrucţiuni , GOSUB şi RETURN. Prin GOSUB se apelează subrutina, iar prin RETURN se revine din subrutină în program.

Instrucţiunea GOSUB provoacă lansarea în execuţie a unei subrutine.

Formatul general este:

GOSUB nr. linie

unde „nr. linie" este numărul primei linii din subrutină.

GOSUB se comportă ca o instrucţiune GO TO cu excepţia faptului că memorează şi numărul de ordine al liniei imediat următoare ei, de unde se va relua programul după execuţia totală a subrutinei.

Instrucţiunea RETURN realizează revenirea în program şi marchează sfîrşitul subrutinei.

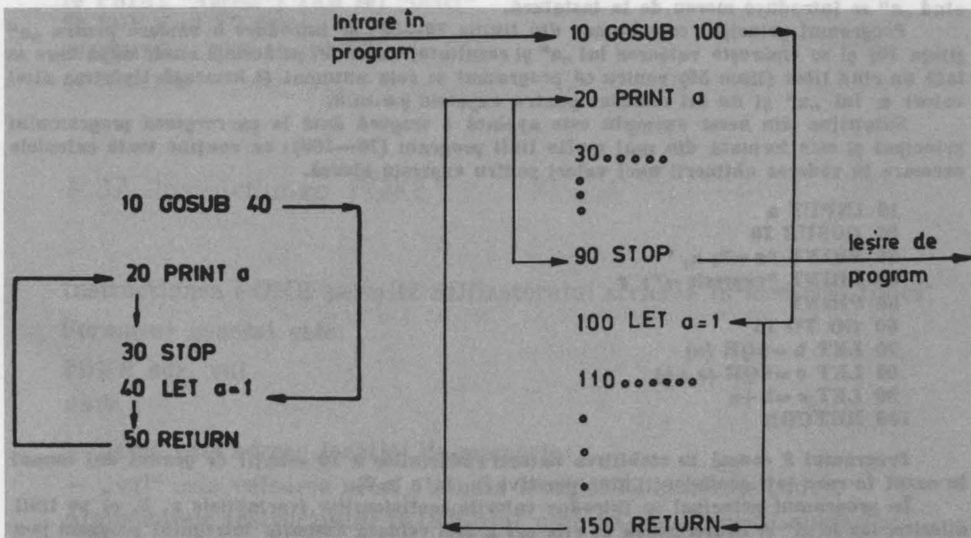


Fig. 9.53. Fig. 9.54.

Formatul general este:

RETURN

(fără nici un argument).

Liniile program cuprinse între linia cu numărul de ordine specificat în GOSUB şi linia RETURN alcătuiesc subrutina.

Cînd controlul este transferat unei subrutine ea se execută pînă la întîlnirea unui RETURN, după care se revine la linia imediat următoare unui GOSUB.

Transferul controlului generat de GOSUB şi RETURN, în cazul unui program ce afişează valoarea unei variabile care este furnizată de o subrutină simplă (formată numai dintr-o singură linie program, linia cu numărul de ordine 40) se vizualizează prin schema (figura 9.53)

O subrutină formată din mai multe linii program se vizualizează ca în figura (9.54)

Nu există posibilitatea declarării explicite a unei subrutine.

Se spune că o subrutină nu are un nume şi nu este recunoscută prin el, aşa cum se întîmplă în alte limbaje de programare, ci numai prin numărul specificat în instrucţiunea GOSUB; numărul acesta reprezintă numărul de ordine al primei linii program a subrutinei, iar sfîrşitul este considerat ca fiind prima linie program întîlnită în parcurgerea secvenţială şi crescătoare, ce conţine o instrucţiune RETURN.

Subrutinele sînt recursive (se pot folosi de mai multe ori în programul în care au fost definite). Un program poate conţine mai multe subrutine. O subrutină poate fi plasată oriunde în corpul programului.

EXEMPLE:

Programul 1 calculează valoarea expresiei

$$\sqrt{a}+\sqrt{a+\sqrt{a}}$$

cînd „a" se introduce mereu de la tastatură.

Programul principal este format din liniile 10—50; se introduce o valoare pentru „a" (linia 10) şi se tipăreşte valoarea lui „a" şi rezultatul expresiei pe acelaşi rînd, după care se lasă un rînd liber (linia 50) pentru că programul se reia automat şi urmează tipărirea altei valori a lui „a" şi un alt rezultat pentru expresie ş.a.m.d.

Subrutina din acest exemplu este apelată o singură dată la parcurgerea programului principal şi este formată din mai multe linii program (70—100); ea conţine toate calculele necesare în vederea obţinerii unei valori pentru expresia aleasă.

```
 10 INPUT a
 20 GOSUB 70
 30 PRINT "a="; a,
 40 PRINT "expresie="; r
 50 PRINT
 60 GO TO 10
 70 LET b=SQR (a)
 80 LET c=SQR (a+b)
 90 LET r=b+c
100 RETURN
```

Programul 2 constă în stabilirea naturii rădăcinilor a 10 ecuaţii de gradul doi numai în cazul în care toţi coeficienţii sînt pozitivi (a, b, c $\geqslant$ 0).

În programul principal se introduc valorile coeficienţilor (variabilele a, b, c) pe linii diferite, iar în „i" se contorizează de cîte ori a fost reluată execuţia întregului program (s-a stabilit iniţial că de 10 ori) şi se tipăresc diferite mesaje potrivit semnului discriminantului (valoarea expresiei b^2—4·a·c).

Subrutina se apelează de trei ori la o singură parcurgere a programului principal şi este alcătuită din mai multe linii program (liniile 80—100), prin ea executîndu-se tipărirea coeficienţilor ecuaţiilor.

```
10 LET i=0
20 LET i=i+1
30 IF i > 11 THEN STOP
40 INPUT a'b'c
50 LET d=b ↑ 2—4 * a * c
60 IF d > 0 THEN GOSUB 90 : PRINT „rad. reale distincte“: GO TO 20
70 IF d=0 THEN GOSUB 90 : PRINT "rad. reale egale": GO TO 20
80 GO SUB 90: PRINT "rad. complexe": GO TO 20
90 PRINT "a="; a; "b="; b; "c="; c
100 PRINT
110 RETURN
```

9.36. Instrucţiunea PEEK

Instrucţiunea PEEK oferă utilizatorului accesul la memoria fizică.

Formatul general este:

PEEK adr

unde „adr“ este o valoare ce reprezintă adresa locaţiei de memorie ce urmează să fie citită.

EXEMPLU:

Programul următor vizualizează primii 21 octeţi din memoria ROM şi adresele lor.

```
10 PRINT "Adresa"; TAB 10; "Octet"
20 FOR a=0 TO 20
30 PRINT a TAB 10; PEEK a
40 NEXT a
```

9.37. Instrucţiunea POKE

Instrucţiunea POKE permite utilizatorului scrierea în memoria fizică.

Formatul general este:

POKE adr, val

unde

— „adr“ este adresa locaţiei de memorie;

— „val“ este valoarea ce se atribuie locaţiei de memorie „adr“.

Argumentele instrucţiunii POKE sînt numere sau expresii numerice. Instrucţiunile POKE şi PEEK sînt complementare: prima introduce în memorie o valoare, a doua extrage din memorie acea valoare.

EXEMPLU:

Linia

10 POKE 31000,57

încarcă la adresa 31000 valoarea 57, iar linia

20 PRINT PEEK 31000

afişează valoarea 57 găsită în memorie la adresa 31000.

Un alt exemplu de utilizare se referă la faptul că sunetele din gama medie sînt cele mai plăcute la redare, iar sunetele grave se percep ca nişte păcănituri. Ele se pot prelungi cu ajutorul instrucţiunii POKE pentru a deveni mai naturale, astfel:

POKE 23609, m
unde m=0, ... 255.

9.38. Instrucţiunea FLASH

Calculatorul HC-85 oferă diferite facilităţi în ceea ce priveşte culoarea ecranului sau a modului de afişare. Pe un televizor alb negru culorile corespund unor tonuri de gri ordonate de la închis la deschis. Orice caracter are asociate 2 culori: culoarea caracterului propriu-zis şi culoarea fondului.

La pornirea calculatorului sistemul lucrează cu caractere negre pe ecran alb. Tipărirea poate fi făcută normal, dar există şi posibilitatea să apară pe ecran pîlpîind. Pîlpîirea se obţine inversînd continuu culoarea caracterului cu culoarea fondului.

Instrucţiunea FLASH realizează pîlpîirea.

Formatul general este:

FLASH nr.

unde „nr" este 0 sau 1 (0 pentru normal, 1 pentru pîlpîire).

9.39. Instrucţiunea PAPER

Culorile realizate cu HC-85 sînt în număr de 8 şi sînt numerotate de la 0 la 7 conform tastelor numerice pe care se găsesc.

Lista culorilor este următoarea:

0 — negru
1 — albastru
2 — roşu
3 — mov (magenta)
4 — verde
5 — bleu
6 — galben
7 — alb

Atributele de culoare sînt asociate unui caracter, deci ele corespund celor 8×8=64 puncte (pixeli). Nu este posibil ca într-un caracter să se găsească mai mult de două culori (culoarea fondului şi culoarea cu care se scrie).

Valorile acestor atribute pot fi modificate pe parcursul execuţiei unui program.

Instrucţiunea PAPER stabileşte culoarea fondului (sau culoarea hîrtiei).

Formatul general este:

PAPER nr.

unde „nr." este un număr cuprins între 0 şi 9.

Cu numerele de la 0 la 7 se stabileşte culoarea dorită.

Folosirea numărului 8 face ca fondul să ia culoarea caracterului anterior tipărit, iar numărul 9 indică contrastul dintre fond şi ceea ce se afişează.

9.40. Instrucţiunea INK

Instrucţiunea INK stabileşte culoarea caracterelor care se afişează pe ecran (sau culoarea cernelii).

Formatul general este:

INK nr.

unde „nr." este un număr cuprins între 0 şi 9.

Folosirea numerelor de la 0 la 7 şi 9 au aceeaşi semnificaţie ca pentru PAPER. În cazul folosirii numărului 8 caracterul ia culoarea fondului.

EXEMPLE:

Programul 1. Desenează spaţii divers colorate prin folosirea instrucţiunilor PAPER şi INK.

```
20 FOR n=1 TO 10
30 FOR c=0 TO 7
40 PAPER c: PRINT "   "; : REM spaţii colorate
50 NEXT c: NEXT n
60 PAPER 7
70 FOR c=0 TO 3
80 INK c: PRINT c; "   ";
90 NEXT c: PAPER 0
100 FOR c=4 TO 7
110 INK c: PRINT c; "   ";
120 NEXT c
130 PAPER 7: INK 0
```

Programul 2

```
10 INK 9
20 FOR c=0 TO 7
30 PAPER c
40 PRINT c
50 NEXT c
```

face să contrasteze culoarea cernelii cu cea a hîrtiei; cînd una are culoarea negru, albastru, roşu sau mov cealaltă are culoarea verde, bleu, galben, alb.

Programul 3 se obţine prin completarea programului 2 cu liniile:

```
60 INK 9
70 PAPER 8
80 PRINT AT 0, 0; FOR n=1 TO 100
90 PRINT n
100 NEXT n
```

În această situaţie, culoarea cernelii este făcută mereu să contrasteze cu vechea culoare pe care o avea fondul, în fiecare poziţie de caracter.

9.41. Instrucţiunea INVERSE

Instrucţiunea INVERSE inversează culoarea fondului cu culoarea cernelii. Datorită acestui efect INVERSE este folosită la ştergerea unor puncte, drepte etc. (atenţie la respectarea direcţiei şi sensului) prin redesenarea acestora cu aceeaşi culoare ca cea a fondului ecranului.

Formatul general este:

INVERSE 1

EXEMPLU:

Utilizînd programul

```
10 PLOT 0, 0
20 DRAW 255, 175
30 PLOT 0, 0
40 DRAW INVERSE 1; 255, 175
```

se constată că diagonala trasată de liniile 10 şi 20 este ştearsă prin suprapunere cu diagonala trasată de liniile 30 şi 40 care are aceeaşi culoare cu fondul.

9.42. Instrucţiunea BORDER

Instrucţiunea BORDER stabileşte culoarea marginii ecranului.

Formatul general este:

BORDER nr.

unde „nr." este un număr cuprins între 0 şi 7.

Marginea ecranului nu are posibilitatea de pîlpîire şi este de luminozitate normală.

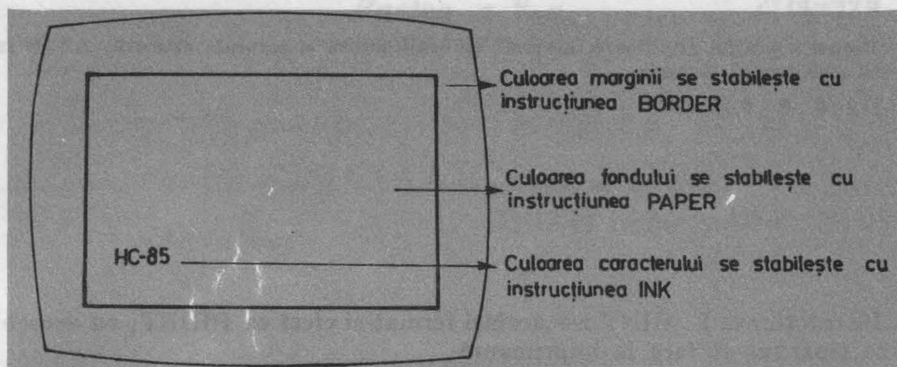


Fig. 9.55.

9.43. Instrucțiunea POINT

Instrucţiunea POINT arată dacă un pixel are culoarea cernelii sau culoarea fondului.

Formatul general este:

POINT (x, y)

unde „x" şi „y" sînt coordonatele unui punct.

Instrucţiunea furnizează valorile:

a) 0 dacă punctul are culoarea fondului (PAPER);
b) 1 dacă punctul are culoarea cernelii (INK).

9.44. Instrucțiunea ATTR

Instrucţiunea ATTR furnizează un număr în care sînt înglobate atributele unui caracter.

Formatul general este:

ATTR (x, y)

unde „x" şi „y" sînt coordonatele unui punct.

Instrucţiunea furnizează o valoare care este suma a patru numere după cum urmează:

a) 128 dacă poziţia pîlpîie, 0 dacă este stabilă;
b) 64 dacă poziţia este strălucitoare, 0 dacă este normală;
c) 8 * n unde n este culoarea fondului;
d) m unde m este culoarea cernelii.

EXEMPLU:

Pentru o poziţie pîlpîitoare, normală cu fond galben şi cerneala albastră, ATTR furnizează următoarea valoare:

128 +0 +8 * 6 +1 =177

9.45. Instrucţiunea LPRINT

Instrucţiunea LPRINT are acelaşi format şi efect ca PRINT, cu deosebirea că tipărirea se face la imprimantă.

9.46. Instrucţiunea LLIST

Instrucţiunea LLIST are acelaşi format şi efect ca LIST, cu deosebirea că listarea se face la imprimantă.

9.47. Instrucţiunea COPY

Instrucţiunea COPY tipăreşte la imprimantă o copie a ecranului.

LIST urmat de COPY (=LLIST) generează listarea la imprimantă. Tipărirea la imprimantă poate fi oprită în orice moment prin acţionarea tastei BREAK.

COPY nu are efect în cazul listărilor automate (o listare automată se obţine tastînd CR după sfîrşitul programului).

Partea a V-a

PROGRAMAREA ÎN LIMBAJUL LOGO PE CALCULATORUL HC-85

Capitolul 10. Caracteristicile şi elementele limbajului LOGO

În acest capitol se prezintă cîteva aspecte prin care se arată că LOGO este un limbaj adecvat pentru utilizarea calculatoarelor în procesul de predare-învăţare. În legătură cu calculatoarele, aproape toţi oamenii au auzit de viteza şi precizia cu care acestea execută şiruri lungi de operaţii complicate, obositoare pentru om. Retribuţia, telefonul, plata energiei electrice consumate, etc., ne vin de la calculator pe bucăţi de hîrtie imprimată.

Se pare că putem accepta că, azi, calculatoarele fac parte din viaţa noastră cotidiană. Dar nu toţi sînt mulţumiţi în situaţia aceasta, mai ales atunci cînd calculatoarele, dar mai ales operatorii şi programatorii acestora comit erori. Trebuie înţeles însă un lucru fundamental şi anume că utilizarea calculatoarelor este o activitate creativă desfăşurată de om.

Orice activitate creativă implică gîndire productivă, rezolvare de probleme, imaginaţie. Cu ajutorul calculatoarelor activitatea creativă cîştigă un plus de utilitate, originalitate şi adaptare la realitate.

Calculatorul face ceea ce face, pentru că cineva l-a instruit ce să facă şi cum să facă. Cine poate să-şi exprime gîndurile în limbajul calculatorului a dobîndit o nouă cale de a gîndi, scrie şi comunica, a descoperit noi modalităţi de rezolvare a problemelor. Apariţia calculatoarelor personale, accesibile şi nespecialiştilor în domeniu, este adesea comparată cu apariţia materialelor tipărite, cu cîteva secole în urmă. Impactul major pe care tehnologiile tipografice l-au avut asupra vieţii de zi cu zi s-a produs abia după alfabetizarea marii majorităţi a populaţiei. Mulţi, altfel bine pregătiţi în diferite domenii, consideră calculatoarele nişte maşini sofisticate şi ostile comunicaţiei directe, cu care se poate comunica numai prin intermediul unor „învăţaţi" (care ştiu să „scrie" şi să „citească" în limbajul calculatorului). Realitatea este însă alta.

Milioane de oameni pot azi comunica direct cu calculatoarele, descoperind noi utilizări ale acestora, pe care nu și le-ar fi putut imagina înainte. Dar ca și lumea înconjurătoare, calculatorul este ceea ce face, nu ceea ce spunem sau credem despre el. Nimic nu poate fi mai convingător decît „făcînd, văzînd și crezînd ceea ce se vede". Pentru aceasta este necesară învățarea unui limbaj de programare.

Oamenii au inventat o mare varietate de limbaje de programare.

10.1. Evoluția limbajelor de programare

Apariția și dezvoltarea limbajelor de programare este strîns legată de evoluția structurii și arhitecturii calculatoarelor. Desigur că viteza de efectuare a operațiilor elementare este un indicator de performanță important, dar din punctul de vedere al limbajelor de programare, capacitatea memoriei interne a calculatorului este mai importantă. Datorită costului ridicat și a posibilităților tehnologice din perioada respectivă, calculatoarele din primele generații aveau o memorie internă de capacitate redusă.

Aceasta a impus ca limbajele de programare să fie simple pentru calculator, chiar cu prețul ca acestea să fie greu de utilizat, neprietenoase pentru programatori. Astfel, programatorul trebuia să țină seama de o serie de restricții și precizări privind numele variabilelor utilizate, tipul, structura și dimensiunea datelor prelucrate. Prin convenții implicite (numele variabilelor întregi să înceapă cu una din literele I, J, K, L, M, N sau numele variabilelor de tip șir să se termine cu caracterul $, variabilele de tip tablou să fie formate din cel mult două caractere începînd cu o literă) și prin declarații explicite compilatorul poate determina numărul de celule de memorie necesar fiecărei variabile, facilitînd astfel alocarea și gestiunea spațiului de memorie limitat. Există și păreri că aceste constrîngeri sînt de fapt avantaje prin faptul că obligînd programatorul să declare tipurile de date ale variabilelor, dimensiunea structurilor de date utilizate, etc., îl obligă pe acesta să fie mai ordonat, mai meticulos în conceperea și scrierea programelor. După alte păreri, aceste neplăceri constituie ceva intrinsec activității de programare.

Mai tîrziu, pe la sfîrșitul anilor 1970, odată cu apariția calculatoarelor personale s-a lărgit considerabil accesul la facilitățile oferite de tehnica de calcul. S-a accelerat astfel schimbarea relației noastre față de cultură, constituită în principal din artă și literatură, avînd un caracter descriptiv, static. Știința în general, informatica în special, oferă posibilitatea unor reprezentări ale lumii înconjurătoare, mai dinamice și predictive, utilizînd legi și modele specifice. Informatica, la fel ca matematica, are o parte de interes comun, dar în același timp fiecare ramură a științei trebuie să-și dezvolte propria ramură a informaticii. Se impune astfel o pregătire de bază, fundamentală în domeniul informaticii utilizînd calculatoarele ca mijloace de realizare a acestui scop.

Trebuie înțeles că însușirea unor „scheme" sau „rețete" de programare nu înseamnă educație în calculatoare. Deși impactul calculatoarelor personale este uriaș în crearea a ceea ce am putea numi o nouă ramură a culturii universale, informatica, trebuie să observăm că acestea au promovat confuzia că ceea ce

este simplu pentru calculator este simplu şi pentru utilizatori. Astfel, limbajul BASIC, care constituie „limba maternă" a calculatoarelor personale este uşor de implementat chiar pe calculatoarele personale cu memorie foarte mică (8—16 Kocteţi). Avînd un vocabular relativ redus de cuvinte cheie BASIC este uşor de învăţat, dar extinderea acestuia cu noi cuvinte cheie, care corespund mai bine necesităţilor programatorului la un moment dat, este dificilă. Puterea de expresie scăzută a limbajului îl fac greu de utilizat şi nu oferă facilităţi deosebite programatorilor începători pentru a scrie programe simple care fac lucruri interesante (spectaculoase).

Pe de altă parte, oamenii au inventat peste o sută de limbaje de programare şi se poate spune că într-un anumit sens toate sînt la fel. Fiecare are un vocabular de aproximativ o sută de cuvinte şi simboluri. Fiecare limbaj este bazat pe o gramatică strict definită, fără excepţii. „Propoziţiile" dintr-un limbaj pot fi uşor traduse în „propoziţii" similare dintr-un alt limbaj.

Astfel, dacă o problemă poate fi rezolvată într-un limbaj de programare oarecare, ea poate fi rezolvată cumva şi în alt limbaj. La ora actuală cei mai mulţi programatori „vorbesc" în BASIC cu calculatoarele lor, datorită răspîndirii calculatoarelor personale. Dar cele mai multe linii de program s-au scris, probabil, în COBOL, un limbaj adecvat aplicaţiilor economice.

Marea majoritate a programelor ştiinţifice şi inginereşti sînt scrise fără îndoială în FORTRAN, unul dintre cele mai vechi limbaje de programare a calculatoarelor. Cînd IBM, cea mai mare firmă producătoare de calculatoare, anunţa lansarea limbajului PL/1, s-a spus că acesta o să înlocuiască FORTRAN-ul şi COBOL-ul deoarece face tot ce fac acestea, şi chiar mai mult. Cei care lucrează în zona inteligenţei artificiale (IA) nici nu concep utilizarea altui limbaj decît LISP sau PROLOG.

Esenţial este însuşirea unui mod de a comunica cu calculatorul, indiferent de limbajul utilizat. Nu se pune problema de a găsi un limbaj „bun" care să le înlăture pe cele „rele". Limba Esperanto este mai universală decît limba română, dar Eminescu a scris în limba română ceea ce nimeni nu a scris în Esperanto. Diferenţele dintre limbajele de programare sînt relativ mici dar suficiente pentru a justifica o anumită opţiune. Diferenţa principală dintre diferite limbaje constă în faptul că un anumit tip de probleme se rezolvă mai uşor într-un limbaj decît în altele. Dar care trebuie să fie facilităţile unui limbaj adecvat rezolvării problemelor specifice procesului de învăţare?

În continuare sînt prezentate cîteva dintre acestea:

— să fie simplu şi puternic în acelaşi timp, astfel ca programe simple să facă lucruri interesante, stimulîndu-i astfel pe începători;

— să ofere facilităţi pentru învăţarea experimentală, prin descoperire, mai bine corelată cu gîndirea intuitivă;

— să ofere posibilitatea utilizării puterii de calcul pentru reformularea unor idei tradiţionale din matematică şi alte ştiinţe, pentru a le face mai accesibile oamenilor cu aptitudini matematice dintr-o gamă mai largă;

— să faciliteze controlul individual asupra unor resurse de calcul puternice, utilizate ca unelte universale pentru învăţare, recreere şi explorare;

— să faciliteze dezvoltarea unor deprinderi în rezolvarea problemelor în general;

— să faciliteze însuşirea principiilor programării structurate;

— să constituie un mediu de învăţare pentru anumite discipline: matematică, fizică, chimie, biologie, literatură, muzică, etc.

— să ofere posibilitatea de a învăţa celor care, dintr-un motiv sau altul nu au succes în clase cu program tradiţional;

— să ofere o bază pentru un tip nou de şcoală, bazată pe principiile Piagetiene de predare-învăţare (vezi glosarul de termeni din paragraful 10.10) utilizînd calculatoarele ca elemente universale în procesul de învăţare;

În continuare se va prezenta LOGO, unul dintre limbajele care răspund în mare măsură acestor cerinţe.

10.2. Ce este LOGO? Caracteristici.

În general prin LOGO se înţelege denumirea unui limbaj de programare. Creatorii acestui sistem, denumit LOGO, afirmă însă că aceasta este denumirea unei filozofii privind educaţia şi a unei familii de limbaje de programare în continuă evoluţie care constituie suportul pentru implementarea acestei filozofii.

LOGO este un limbaj pentru învăţare în general. Mai concret LOGO este pe de o parte, un limbaj de învăţare a programării calculatoarelor, iar pe de altă parte, un limbaj de învăţare a unui mod de gîndire.

Istoric vorbind LOGO îşi are rădăcinile atît în cercetările din ştiinţa calculatoarelor, în special în domeniul inteligenţei artificiale, cît şi în studiile şi cercetările lui Jean Piagèt privind dezvoltarea aptitudinilor de a gîndi la copii.

LOGO a fost iniţial creat de firma Bott, Beranek & Newman, în Cambridge, Massachusetts în anul 1968. De atunci a cunoscut o continuă dezvoltare prin cercetările efectuate la Laboratorul de Inteligenţă Artificială şi la Departamentul de Studii şi Cercetări în Educaţie de la Institutul de Tehnologie din Massachusetts (MIT), precum şi alte universităţi din întreaga lume. Cercetările au fost efectuate utilizînd cele mai puternice calculatoare disponibile în acea perioadă.

În 1979 grupul LOGO de la MIT a început activitatea de implementare a limbajului pe calculatorul personal TI99/4 sub conducerea lui Seymour Papert şi pe calculatorul personal Apple II sub conducerea lui Harold Abelson. Limbajul a fost apoi implementat şi pe alte calculatoare personale. Nu trebuie să se facă confuzia între limbaj şi una din aceste implementări supuse unor restricţii datorită limitelor performanţelor calculatoarelor personale.

Dar pentru a răspunde mai bine la întrebarea, „Ce este LOGO“, vor fi arătate în continuare, principalele caracteristici prezentate de LOGO privit ca limbaj de programare a calculatoarelor.

LOGO este un limbaj procedural. Programele în LOGO sînt create prin asocierea comenzilor în grupuri denumite proceduri şi utilizarea procedurilor drept comenzi în alte proceduri, procesul putînd continua pe oricîte nivele. Fiecare pas într-o procedură poate fi comandă elementară din LOGO sau o procedură definită de utilizatori. Aceasta oferă posibilitatea implementării (şi a însuşirii) unei strategii foarte puternice de rezolvare a problemelor: spargerea problemei în părţi din ce în ce mai mici şi scrierea unei proceduri pentru fiecare parte. Procedurile pot comunica între ele prin intrări şi ieşiri.

LOGO este interactiv. Orice comandă LOGO fie că este predefinită în limbaj sau este definită de programatori ca o procedură, poate fi executată prin simpla introducere de la tastatură a numelui acesteia.

Introducerea, ştergerea sau schimbarea unei linii de program este simplă.

Editorul integrat în LOGO facilitează definirea, execuţia sau modificarea procedurilor fără a utiliza programe separate de încărcare, editare, compilare, etc. Eficienţa unui limbaj diferă pentru limbajele interactive (LOGO, BASIC, LISP, APL), faţă de cele neinteractive (FORTRAN, PASCAL, MODULA, C).

Dezvoltarea şi punerea la punct a programelor este mai eficientă în cazul limbajelor interactive, dar un program pus la punct se execută, mai eficient în cazul limbajelor neinteractive.

Orice program scris într-un limbaj de nivel înalt (interactiv sau neinteractiv) trebuie tradus într-un limbaj specific calculatorului, denumit limbaj maşină. Această traducere este efectuată de un alt program denumit interpretor pentru limbajele interactive şi compilator pentru cele neinteractive.

Compilarea este un proces complex, care durează mult, dar programul rezultat, cînd este corect, este obţinut într-o formă binară „pură" (în limbaj maşină) şi execuţia lui ulterioară este rapidă (nu mai este nevoie de compilator). Prin contrast am putea spune că interpretorul face această traducere linie cu linie, de fiecare dată cînd se execută programul. De fapt, interpretorul nu traduce programul în formă binară ci el însuşi execută paşii necesari în limbaj maşină pentru fiecare pas din programul în limbaj superior.

Cele mai multe interpretoare, inclusiv cel pentru LOGO, produc (la prima execuţie) o formă intermediară (o traducere parţială) care face ca execuţiile ulterioare să fie mai rapide.

Limbajele interpretate pot fi interactive. Limbajele interactive oferă şi facilităţi de depanare interactivă a programelor ceea ce duce la creşterea productivităţii programatorului. De altfel pentru specificul activităţii de învăţare (a programării) un program care nu mai are pene este neinteresant. În general este abandonat şi se demarează un nou proiect.

LOGO este un limbaj recursiv. Într-un limbaj procedural o procedură poate apela o altă procedură (denumită subprocedură) pentru a efectua o parte din sarcinile sale. Un limbaj este recursiv dacă permite ca o procedură să fie subprocedură pentru ea însăşi. Puterea de expresie a acestui concept constă în faptul că permite ca o problemă complicată să fie descrisă în termenii unei versiuni mai simple a ei însăşi ceea ce conduce la posibilitatea de descriere (în limbaj recursiv) într-o formă foarte compactă a unor probleme foarte complexe. Recursivitatea este unul dintre conceptele greu de înţeles din programare, asupra căruia se va reveni, cu exemplificări, în capitolul următor.

Logo este extensibil. Într-un anumit sens, toate limbajele procedurale sînt extensibile deoarece ele permit extinderea facilităţilor limbajului cu noi operaţii, pentru care s-au scris procedurile adecvate.

În mod strict însă, un limbaj este extensibil dacă procedurile definite de utilizator „arată la fel" ca procedurile primitive, predefinite ale limbajului.

LOGO prezintă o singură abatere de la această regulă de extensibilitate. Acesta constă în faptul că unele operaţii aritmetice primitive (adunare, scădere, ...) pot fi scrise în forma infixată, cu operatorul între cei doi operanzi (Ex: 5+3), iar operaţiile noi, definite de utilizator (prin proceduri adecvate), trebuie neapărat să fie în forma prefixată, operatorul precede operanzii (Ex: SUM 5 3). Asupra modului în care se scrie şi se citeşte o linie în LOGO se va reveni în capitolul următor.

În ceea ce priveşte obiectele reprezentînd date (acele entităţi care pot fi denumite cu nume de variabile independente, pot fi transmise ca intrări sau ieşiri ale unor proceduri), LOGO are cîteva caracteristici distinctive.

LOGO nu este orientat pe tipuri de date. În cele mai multe limbaje tipul unui obiect reprezentat printr-o variabilă trebuie specificat explicit prin declaraţii sau este specificat implicit prin restricţii în alcătuirea numelui variabilei (Ex: — să înceapă cu una dintre literele I-N pentru a reprezenta un întreg în FORTRAN, să se termine cu $ pentru a reprezenta un şir în BASIC). În LOGO ca şi în LISP (din care este de fapt inspirat) variabilele nu au asociat un anumit tip. Orice variabilă poate lua orice valoare. Aceeaşi variabilă poate reprezenta un întreg la un moment dat şi un cuvînt (un şir de caractere) la alt moment de timp dat. Iniţial variabilele au fost tipizate pentru a uşura munca de scriere a compilatoarelor care trebuie să stabilească corespondenţa dintre funcţiile primitive din limbajul de nivel înalt şi cele din limbaj maşină. Deoarece în limbaj maşină adunarea unor întregi se face cu altă operaţie decît adunarea numerelor reale, iar operaţiile în simplă precizie sînt diferite de cele în precizie extinsă, traducerea în limbaj maşină este mai simplă dacă limbajul superior obligă la aceleaşi restricţii. Mai tîrziu, filozofînd pe marginea limbajelor de programare, s-a emis ideea că tipizarea datelor are marele avantaj că obligă programatorul să fie mai ordonat în utilizarea variabilelor (o variabilă este utilizată pentru un singur scop într-un program).

LOGO realizează folosirea disciplinată a variabilelor, prin utilizarea unei variabile pentru un singur scop într-o manieră diferită.

LOGO este un limbaj procedural în care variabilele (netipizate) sînt asociate unei proceduri şi nu întregului program. Asupra acestui aspect se va reveni în capitolul următor.

LOGO permite prelucrarea listelor. Limbajele de nivel înalt şi unele limbaje de asamblare mai evoluate permit gruparea datelor şi structurilor de complexitate mai ridicată (tablouri, înregistrări, structuri). În limbajele BASIC şi FORTRAN de exemplu se pot defini tablouri de variabile prin declaraţii în care se specifică numele şi dimensiunea acestora. În LOGO se pot defini liste, ca modalitate principală de grupare a datelor în structuri complexe, structurile de date din LOGO fiind inspirate din LISP.

Listele se deosebesc de tablouri atît prin structură cît şi prin conţinut. Tablourile sînt structuri fixe, statice cu dimensiunile fixate prin declaraţii la începutul programului. Prin contrast, listele au o structură dinamică, structura lor nu trebuie specificată iniţial, ci poate) evolua după necesităţi, în timpul execuţiei programului. În ceea ce priveşte conţinutul, tablourile au un conţinut omogen (conţin numai date de acelaşi tip). Pot fi definite tablouri de numere întregi, şi numere reale, de informaţii alfanumerice, dar separat. În acelaşi tablou nu pot fi definite date de tipuri diferite. În Logo listele nu sînt tipizate. Fiecare element dintr-o listă poate conţine orice obiect acceptat de limbaj, un număr (întreg sau real), un cuvînt (şir de caractere alfanumerice) sau o altă listă (şi aceasta pe oricîte nivele). Listele oferă o flexibilitate mult mai mare în structurarea datelor după necesităţile fiecărui proiect, dar accesul la un element oarecare al structurii se face mult mai uşor la tablouri. Cunoscînd adresa de început a tabloului şi faptul că fiecare element ocupă acelaşi număr de celule de memorie se ajunge imediat la elementul dorit. Memorarea listelor, avînd în vedere că au o structură dinamică şi că fiecare element poate ocupa un număr diferit de celule de memorie, se face într-o manieră mai complicată. Pentru a ajunge la un element oarecare din listă, aceasta trebuie parcursă element cu element, într-un sens sau altul. Căutarea unui element într-o listă este mai lentă, dar nu este mai dificilă pentru programator decît căutarea într-un

tablou. Alte limbaje, cum ar fi PASCAL, MODULA, C, nu pot implementa liste direct, dar cu ajutorul variabilelor de tip indicator (pointer) se pot prelucra structuri de tip listă. Din considerente metodologice am privit LOGO din două puncte de vedere: limbaj de programare şi limbaj de învăţare. Ca limbaj adecvat procesului de predare — învăţare LOGO are cîteva caracteristici distincte dintre care vor fi prezentate cîteva în continuare.

Este bazat pe ideea lui Jean Piaget că pînă şi copiii (chiar mici) au teorii. Procesul de predare-învăţare nu este o chestiune de a fi bun sau rău. Învăţarea şi predarea se împletesc într-un proces continuu de „depanare“, de umplere a golurilor în cunoaştere de îndată ce sînt descoperite. LOGO facilitează învăţarea prin descoperire. Penele din programele în LOGO nu trebuie privite neapărat ca greşeli; analizate, acestea pot sugera sau chiar pot duce la rezultate mai interesante decît obiectivele propuse iniţial. Prin analiza empirică a sute de proiecte de utilizare a calculatoarelor în şcoli s-a ajuns la concluzia ca prin învăţarea şi exersarea unui limbaj de programare (în special LOGO) elevii dobîndesc deprinderi cognitive cum ar fi:

— gîndire riguroasă şi posibilitate de exprimare cu acurateţe şi precizie a ideilor;

— însuşirea unor metode euristice cum ar fi planificarea activităţilor, decompoziţia unei probleme în subprobleme, alcătuirea unor modele de reprezentare, stabilirea unor analogii;

— abilitate în descoperirea şi înlăturarea erorilor din soluţiile la o problemă dată;

— înţelegerea faptului că cele mai multe probleme au mai multe strategii de elaborare a soluţiilor;

— înţelegerea şi posibilitatea de utilizare a unor concepte de bază cum ar fi: procedură, funcţie, variabilă, recursivitate, etc.

Exprimarea în LOGO este clară, explicită. LOGO diferă de LISP, în care îşi are rădăcinile, prin caracteristici rezultate din efortul de a-l face cît mai adecvat procesului de predare — învăţare. Ideile fundamentale din programare sînt explicite. Astfel procedurile sînt împărţite în operaţii şi comenzi. Operaţiile sînt proceduri care calculează (generează) diferite valori transmise la ieşire şi care pot fi apoi preluate de alte proceduri. Spre deosebire de operaţii, comenzile sînt proceduri care nu transmit ceva la ieşire ci au un efect imediat (vizibil, într-un fel), de ex: trasează un segment de dreaptă, afişează pe ecran o valoare, un text, sau modifică valoarea unei variabile. Prin notaţii adecvate se elimină un mare număr de confuzii în identificarea procedurilor, variabilelor şi a valorilor acestora.

Astfel, orice nume format din caractere alfanumerice fără semne speciale reprezintă o procedură (X reprezintă o procedură, „X reprezintă numele variabilei X iar :X reprezintă valoarea variabilei X). Unei variabile i se atribuie o valoare în mod clar, explicit, utilizînd operatorul MAKE care sugerează o schimbare. Astfel X=X+1 pare aberant din punct de vedere algebric. În LOGO aceasta se scrie MAKE „X :X+1 adică variabila cu numele X, notată „X va primi (va fi făcută egală cu) valoarea actuală a lui X, notată :X plus 1.

LOGO este stimulativ. Cele mai multe limbaje de programare suferă prin lipsa facilităţilor de rezolvare cu uşurinţă a unor probleme interesante, chiar de către începători. Astfel, deseori un curs de programare pentru începători începe cu rezolvarea şi discuţia ecuaţiei de gradul doi şi continuă apoi cu operaţii cu matrici, etc.

Unul dintre aspectele cele mai atractive privind programarea în LOGO constă în facilitățile grafice și de prelucrare a textelor în limbaj natural. Utilizînd un instrument conceptual denumit *penel* (cu sensul cunoscut pentru cei ce iubesc pictura, sau cu sensul de *peniţa el*ectronică pentru cei fascinați de tehnică) și operațiile și comenzile de lucru cu penelul, se pot obține desene complexe, se pot rezolva probleme de geometrie sau se pot simula fenomene și legi din fizică într-o manieră directă și atractivă. Limbajele de programare consideră în general textele ca șiruri de caractere. În LOGO, literele, cuvintele și propozițiile sînt obiecte cu o ierarhie naturală, ceea ce oferă facilități interesante de analiză a limbajului uman. Între multiplele proiecte și experimente privind utilizarea lui LOGO în școli, prezentate în literatura de specialitate se află și lucrul cu diferite categorii gramaticale. O elevă de 13 ani încercînd să scrie un program care compune poezii a făcut o descoperire interesantă, exprimată astfel: abia acum am înțeles „de ce avem substantive și verbe".

Deși nu părea să aibă dificultăți în lucrul cu categoriile logice avea probleme în înțelegerea gramaticii, a diferenței dintre substantive verbe și adverbe.

Nu înțelegea de fapt la ce este bună gramatica. Încercînd să scrie programul a constatat că ea însăși clasifică cuvintele în categorii, nu pentru că cineva i-a cerut acest lucru ci pentru că trebuia.

Ea trebuia să învețe calculatorul să aleagă cuvintele, din clasa corespunzătoare. Ea a înțeles că și cuvintele, ca și lucrurile în general, pot fi plasate în diferite grupuri, mulțimi, în conformitate cu obiectivele sale. Experiența ei a fost profundă și plină de sens. Nu numai că a înțeles gramatica, dar și-a schimbat relația față de aceasta.

Pe parcurs vor fi prezentate mai multe exemple de programe de acest gen.

LOGO este adecvat în diferite etape ale procesului de predare învățare. LOGO este în general privit ca un limbaj pentru copii deoarece cei mai mulți au un model al procesului de învățare, ca un proces în care numai cei mici învață. Desigur, au fost proiecte de grafică cu penelul în LOGO pentru copii de 4 ani, utilizînd o claviatură adecvată, cu butoane mari, marcate cu poze în loc de cuvinte. S-a introdus chiar și conceptul de procedură, existînd butoane de „start memorare" și „stop memorare" pentru definirea unei proceduri. Pentru a defini mai multe proceduri s-au utilizat butoane de diferite culori. Dar profesorul Harold Abelson a utilizat la MIT facilitățile de grafică cu penelul pentru predarea fizicii la facultate pentru a demonstra principii din mecanica newtoniană și din teoria generală a relativității. Deasemenea, LOGO a fost utilizat pentru grupuri de elevi cu inteligența normală sau superioară, dar handicapați în ceea ce privește posibilitatea de comunicare. Calculatoarele pot fi utilizate, atît ca mediu de comunicare în astfel de situații, cît și ca mijloace de obținere a autonomiei în realizarea unor scopuri.

10.3. LOGO în școli

Implementarea limbajului pe cele mai răspîndite calculatoare personale a permis depășirea stadiului de experimente izolate și trecerea la utilizarea sistematică și generală în procesul de învățămînt în diferite țări ale lumii. Dacă oportunitatea utilizării limbajului în școli este demonstrată prin multiple

proiecte şi experimente, modalităţile concrete de implementare sînt diverse şi depind de context.

Calendarul de lucru, organizarea accesului la calculatoare, etc., depind în mare măsură de dotare, de asigurarea cu cadre de specialitate, de modul cum principiile solide ale învăţămîntului tradiţional sînt corelate cu noile posibilităţi oferite de calculatoare. În orice caz, pentru începători vor trebui fixate următoarele idei:

— programarea este un proces de angajare a calculatorului într-o conversaţie utilizînd numai cuvinte dintr-un vocabular restrîns pe care îl înţelege;

— acest vocabular poate fi uşor extins prin definirea de noi proceduri care apoi pot fi utilizate exact ca şi operaţiile şi comenzile primitive din vocabularul iniţial;

— prin definirea de noi proceduri şi denumirea acestora cu noi cuvinte se dă sens acestor cuvinte, care apoi pot fi folosite în definirea altor proceduri, etc;

— extinderea domeniului de cunoaştere al calculatorului prin noi proceduri implică un proces de definire şi depanare a acestora;

— penele descoperite în procesul de punere la punct a programelor trebuiesc bine analizate şi înţelese.

Ele pot constitui idei pentru noi explorări:

— substituindu-se calculatorului sau penelului şi parcurgînd paşii respectivi, programatorul poate ajunge mai repede la detectarea şi înlăturarea penelor;

— primele programe trebuie să fixeze relaţia dintre obiectiv şi program, apoi dintre obiective şi subobiective, proceduri şi subproceduri;

— facilitarea schimbului de idei, informaţii şi experienţe între elevi poate duce la rezultate spectaculoase.

Programele de lucru cu penelul sînt adecvate acestei faze de început. Învăţarea-programării, ca orice proces de învăţare se desfăşoară conform cu un stil propriu care reflectă personalitatea intelectuală a elevului. Calculatorul oferă o mai mare flexibilitate fiecărui elev de a progresa în stilul şi ritmul propriu, respectîndu-i-se personalitatea. Totuşi, observînd anumite similitudini în abordarea programării în LOGO s-au degajat trei stiluri importante de lucru:

— stilul timid, plecînd de la o micro-explorare. Elevii încep cu timiditate încercînd o dată, de două sau mai multe ori aceeaşi comandă pentru a se convinge că aşa este. Ei execută 2+3, apoi vor să se convingă că 3+2 duce la acelaşi rezultat etc. Numai după aceea trec la o explorare planificată, direcţionată de anumite obiective;

— stilul curajos, plecînd de la o macro-explorare. Elevii exersează diferite proceduri, studiază efectele apoi trec la utilizarea acestora în conformitate cu obiective precise;

— stilul organizat, planificat. Elevii abordează orice problemă în mod structural, de jos în sus sau de sus în jos, dar întotdeauna în mod organizat, după un plan riguros.

Aceste stiluri de lucru se regăsesc la orice începător în LOGO, indiferent de vîrstă şi ele trebuie încurajate în mod corespunzător, cunoscînd faptul că ele se întrepătrund în mod natural.

Ce trebuie să ştie un profesor de LOGO?, este o întrebare la care este dificil de răspuns. În orice caz el trebuie să aibă în vedere următoarele:

— trebuie să dispună de o colecţie de proiecte de programare care pun în evidenţă conceptele şi tehnicile de programare în LOGO;

— să dispună de un vocabular adecvat discuţiilor şi dezbaterilor despre programare;

— o subtilă cunoaştere a diferitelor stiluri şi strategii de învăţare a programării, pentru a le putea manipula;

— stăpînirea şi contracararea rezistenţei pe care unii copii sau adulţi o opun lucrului cu categorii matematice şi logice.

În final trebuie accentuate faptul că ideea utilizării calculatoarelor în educaţie nu este susţinută pentru motivul ca fiecare să ştie ceva despre calculatoare, ci faptul că pentru mulţi oameni, programarea calculatoarelor este o modalitate foarte importantă de a învăţa să înveţe, este un mod de explorare intelectuală prin construcţia de modele din ce în ce mai perfecţionate.

Calculatorul oferă toate aceste facilităţi pentru că este o unealtă universală care poate desena, poate să compună muzică sau poezii, sau să conducă maşini unelte sau roboţi sofisticaţi.

Microcalculatoarele produse în ţara noastră HC-85, FELIX PC şi calculatoarele personale compatibile cu APPLE II oferă posibilitatea utilizării limbajului LOGO.

10.4. LOGO pe HC-85

Pentru a utiliza LOGO pe HC-85 este nevoie de:

— calculatorul personal HC-85, Sinclair Spectrum cu 48K sau un calculalator personal compatibil;

— televizor sau monitor alb-negru sau color;

— casetofon şi caseta cu limbajul LOGO sau

— disc magnetic şi discul cu limbajul LOGO.

Încărcarea programului se face normal, iar pe ecran apare un mesaj de bun-venit programatorilor în LOGO.

Apoi, pe ecran, la început de rînd LOGO trimite ? pentru a anunţa utilizatorul că este în regim de aşteptare comenzi de la tastatură. Acest regim corespunde nivelului cel mai de sus, în terminologia LOGO (top level). În continuare se prezintă cîteva aspecte privind utilizarea tastaturii:

ENTER sau CR	— Apăsarea tastei ENTER specifică terminarea introducerii unei linii de program, a unei instrucţiuni etc. (care a început cu?) şi comunică limbajului că poate trece la interpretarea ei. O linie de programare în LOGO poate avea pînă la 250 de caractere, deci mai multe linii pe ecran;
BREAK/SPACE	— Introduce un spaţiu liber;
SS	— Prescurtarea pentru tasta SYMBOL SHIFT;
CS	— Prescurtarea pentru CAPS SHIFT;
CMODE	— Tastînd CAPS şi 2 (în acelaşi timp) tastatura trece în mod „litere mari";
LMODE	— Tastînd CAPS a doua oară tastatura trece în mod „litere mici";
EMODE	— Tastînd CAPS şi SYS tastatura trece în mod editare. În colţul stînga jos al ecranului se afişează caracterul E.

Cînd se apasă o a treia tastă, SYS trebuie ţinută apăsată;

Delete	— Tastînd CS şi 0 se şterge un caracter spre stînga;
←	— Tastînd CS şi 5 se deplasează cursorul cu un caracter spre stînga, fără să se şteargă nimic;
↓	— Tastînd CS şi 6 (în mod editare) se deplasează cursorul în jos cu o linie;
↑	— Tastînd CS şi 7 se deplasează cursorul în sus cu o linie;
→	— Tastînd CS şi 8 se deplasează cursorul cu un caracter spre dreapta;
SS P	— Tastînd SS şi P se afişează apostrof ';
SS Y	— Tastînd SS şi Y se afişează paranteza dreaptă deschisă [;
SS U	— Tastînd SS şi U se afişează paranteza dreaptă închisă];

LOGO tratează în mod diferit parantezele drepte [ ] şi rotunde ( ).

CS — BREAK/SPACE — Tastînd CAPS şi BREAK/SPACE înainte ca LOGO să termine execuţia unei proceduri, forţează întreruperea acesteia şi pe ecran apare mesajul:
— STOPPED!!!
?
şi LOGO intră în regim de aşteptare comenzi (pe nivelul cel mai de sus);

Introducerea de la tastatură a unor proceduri mai simple se poate face direct, utilizînd şi tastele cu semnificaţie specială arătate mai sus.

Pentru editarea unor proceduri complexe definite prin texte lungi se utilizează editorul încorporat în LOGO care facilitează introducerea şi ştergerea/corectarea textelor sub controlul interpretorului LOGO.

Mod editare:

Intrarea în mod editare se face tastînd EDIT sau ED urmat de numele procedurii (sau lista de proceduri) ce urmează să fie editată.

Ex: ED "PROC1 sau ED [CERC ELIPSA POLI]

Dacă există deja o procedură cu acelaşi nume ea va fi afişată pe ecran pentru a fi corectată, actualizată.

Dacă se introduce EDIT sau ED fără un nume de procedură, editorul acţionează asupra textului ce a rămas în zona de memorie rezervată lui, dintr-o editare anterioară.

EDIT [ ] sau ED [ ] lansează întotdeauna editorul cu zona de lucru curată (fără nici un text de la vre-o editare prealabilă).

În mod editare caracterul? (promptul) nu mai apare. Cursorul (pătrat care clipeşte) se deplasează pentru a indica locul unde se va introduce următorul caracter.

Caractere speciale în mod editare:

CS 5	— Mută cursorul la stînga cu un caracter (←);
CS 6	— Mută cursorul în jos cu o linie (↓);
CS 7	— Mută cursorul în sus cu o linie (↑);
CS 8	— Mută cursorul la dreapta cu un caracter (→);
CS 0	— Şterge un caracter spre stînga;
EMODE CS 5	— Mută cursorul la începutul liniei curente;

EMODE CS 6	—	Mută cursorul la sfîrşitul ecranului;
EMODE CS 7	—	Mută cursorul la începutul ecranului;
EMODE CS 8	—	Mută cursorul la sfîrşitul liniei curente;
EMODE B	—	Mută cursorul la începutul textului introdus;
EMODE E	—	Mută cursorul la sfîrşitul textului introdus;
Defilare	—	Dacă textul introdus depăşeşte o pagină de ecran, prin defilare se poate trece la pagina următoare sau la cea precedentă;
SS S	—	Opreşte defilarea. Orice tastă apăsată apoi porneşte defilarea din nou;
EMODE N	—	Mută cursorul la pagina următoare;
EMODE P	—	Mută cursorul la pagina precedentă;
EMODE Y	—	Şterge şi salvează linia în care se găseşte cursorul;
EMODE R	—	Inserează linia salvată cu EMODE Y, în poziţia cursorului;
EMODE C	—	Se iese din mod editare şi se salvează toate modificările efectuate. Editorul afişează un mesaj cu numele procedurii editate;
CS BREAK/ SPACE	—	Se iese din mod editare şi nu se iau în considerare modificările făcute.

În afara modului editare toate tastele cu funcţii speciale de editare pot fi utilizate în cadrul aceleaşi linii de program LOGO.

Editarea numelor de variabile şi a valorii lor se face cu comanda ENDS nume sau ENDS [LISTA de nume]. Fără argument ENDS va afişa toate numele şi valorile lor definite pînă în acel moment.

TO MODE — Tastînd TO se trece în modul TO de definire a unei proceduri. Promptul nu mai este ? ci >. Abandonarea definirii unei proceduri se poate face tastînd CAPS BREAK/SPACE. Pentru corecturi se poate utiliza comanda ERASE de ştergere a procedurii şi apoi se face redefinirea ei. Ultima linie va conţine doar END pentru a marca sfîrşitul definirii unei proceduri. Dacă se definesc mai multe proceduri, fiecare trebuie terminată cu END. La definirea ultimei proceduri cu Editorul nu este necesar să se introducă END. La părăsirea editorului, se introduce implicit END.

10.5. Reguli gramaticale ale limbajului Logo

Plecînd de la setul standard de caractere ASCII şi funcţiile speciale ale tastaturii calculatorului, se pot introduce în calculator şiruri de caractere interpretate în diferite moduri de către programul care le preia de la tastatură.

Un program LOGO este la prima vedere un şir de caractere organizat în linii mai lungi sau mai scurte, aşa cum se vede pe ecranul calculatorului. Dar ca şi în limbajele naturale nu orice şir de caractere formează cuvinte cu sens şi nu orice înşiruire de cuvinte cu sens formează fraze cu sens. Logo înţelege un set de cuvinte introduse de la tastatură care fac parte din vocabularul de bază,

dar şi neologisme al căror sens este transmis calculatorului în cadrul programului.

Deci un program Logo este format din construcţii utilizînd cuvinte din vocabularul predefinit şi construcţii care definesc şi utilizează cuvinte noi îmbogăţind astfel vocabularul. Caracterele şi cuvintele din vocabularul de bază şi regulile de combinare a acestora în propoziţii şi fraze formează gramatica limbajului Logo. Orice program pentru a putea fi interpretat corect de către calculator trebuie scris respectînd cu stricteţe regulile gramaticale care vor fi prezentate în continuare.

În fine, dacă în dialogurile dintre noi uneori mai trecem cu vederea anumite greşeli gramaticale care nu afectează sensul frazelor respective, calculatorul nu iartă nimic. El va semnala imediat orice abatere de la regulile prestabilite pentru a salva interpretări greşite şi generarea unor rezultate fără sens.

Gramatica limbajului Logo are un număr redus de reguli şi nu are excepţii!

10.6. Obiectele limbajului Logo

Datele primite de proceduri la intrare şi datele furnizate la ieşire sînt obiecte sub forma de numere sau şiruri de caractere care în Logo se numesc *cuvinte*. O facilitate deosebită a limbajului Logo constă în posibilitatea de grupare şi prelucrare a datelor în structuri complexe denumite *liste*.

Numerele, cuvintele şi *listele* sînt obiectele cu care operează procedurile în limbajul Logo.

Numere

Numerele sînt reprezentate în limbajul Logo sub *formă de cuvinte* şi pot fi întregi sau fracţionare. De exemplu, 7 este un număr întreg iar 1.21 este un număr fracţionar. Astfel, instrucţiunile următoare au exact acelaşi efect:

```
PR 12                PR "15
12                   15
```

Operaţiile care se pot efectua cu numere şi ordinea în care se execută la evaluarea unor expresii complexe sînt:

— împărţirea, / cu prioritatea cea mai mare;
— înmulţirea, × se execută după împărţire;
— scăderea, — înaintea adunării şi după înmulţire şi împărţire;
— adunarea, + are prioritatea cea mai mică.

Dacă se doreşte o altă ordine de efectuare a operaţiilor se vor utiliza paranteze. Aşa cum s-a arătat la începutul prezentării limbajului, singura excepţie privind modul cum arată procedurile predefinite şi cele definite de utilizator o constituie posibilitatea de utilizare sub forma infixată a operatorilor aritmetici; astfel următoarele forme sînt echivalente:

EXEMPLU:

Forma infixată	Forma standard prefixată
PR 4 + 5	PR SUM 4 5
9	*9*

```
PR  7— 3
4
PR  —5 + 3
—2
PR  —8 —2
—10
PR  —8——2
—6
PR  2 + 3 + 5
10
PR  6 * 3
18
PR  6 * 2 * 5
60
PR  2 + 3 * 4
14
PR  (2 + 3) * 4
20
PR  + 7 / 2
3,5
PR  — 7 / 2
—3,5
PR  7 / 0
Can"t divide by zero
```

```
PR  SUM  7  —3
4
PR  SUM  —5  +3
—2
PR  SUM  —8  —2
—10
PR  SUM  —8  +2
—6
PR  (SUM  2  +3  5)
10
PR  PRODUCT  6  3
18
PR  (PRODUCT  6  2  5)
60
PR  SUM  2 PRODUCT  3  4
14
PR  PRODUCT SUM  2  3  4
20
PR  DIV  7  2
3,5
PR  DIV  —7  2
—3,5
PR  DIV  7  0
Can"t divide by zero,
```

Trebuie observat că în Logo parantezele rotunde sînt utilizate pentru a reprezenta o grupare a unei operaţii cu intrările sale. Asupra regulilor sintactice de utilizare a parantezelor se va reveni în paragrafele următoare.

Cuvinte

Şirurile de caractere sînt denumite *cuvinte* în Logo.

Ca şi numerele care sînt de fapt tot cuvinte, cuvintele în Logo pot fi transferate ca intrări sau ieşiri în cadrul procedurilor. De asemenea se pot efectua operaţii de combinare a unor cuvinte în cuvinte mai lungi sau se poate despărţi un cuvînt în mai multe cuvinte mai scurte. Un cuvînt se prezintă în Logo prin prefixarea acestuia cu caracterul “ .

```
PR  "ELEV
ELEV
```

Numerele şi variabilele logice TRUE şi FALSE sînt cuvinte care se pot scrie fără prefixul “. Astfel următoarele forme sînt echivalente:

```
PR  12          PR  "12
12              12
PR TRUE         PR  "TRUE
TRUE            TRUE
PR  FALSE       PR  "FALSE
FALSE           FALSE
```

Cuvintul care nu are nici un caracter se numeşte cuvînt *vid*.

```
PR  "
```

Un cuvînt începe imediat după caracterul “ sau : şi se termină la primul blanc, [ ] ( ) < > + — × / sau la terminarea liniei. Astfel în exemplul de mai jos caracterul “ de la sfîrşit face parte din cuvînt.

```
PR  "EXEMPLU"
EXEMPLU"
```

Un cuvînt poate conţine orice caracter tipăribil în afară de blanc şi caracterele delimitatoare arătate mai sus, deşi există altă soluţie pentru includerea acestor caractere în cuvinte, dacă este strict necesar. Astfel, un caracter din cele de mai sus dacă este precedat de / adică SYS D este luat ca atare şi nu este interpretat conform funcţiei atribuite. Astfel cuvîntul 3 [A] B poate fi definit ca " 3 / [A] / B pentru a evita interpretarea parantezelor drepte ca delimitatori pentru o listă. Primitivele de prelucrare a cuvintelor sînt prezentate în paragraful următor. Acestea oferă posibilitatea prelucrării caracter cu caracter a şirurilor de caractere care formează cuvinte. Pentru prelucrarea şirurilor de caractere cu structuri complexe se utilizează o reprezentare a acestora sub formă de liste. Să scriem de exemplu o procedură recursivă care prelucrează un cuvînt la fel cum o procedură de numărare inversă prelucrează numerele.

EXEMPLU:

```
TO TRIUNGHI.CUV :CUVÎNT
IF CUVÎNT="THEN STOP
PRINT :CUVÎNT
TRIUNGHI.CUV BUTFIRST :CUVÎNT
END
TRIUNGHI "ABRACADABRA
ABRACADABRA
BRACADABRA
RACADABRA
ACADABRA
CADABRA
ADABRA
DABRA
ABRA
BRA
RA
A
```

Spre deosebire de procedura de prelucrare numere care scade 1 din număr la fiecare execuţie, TRIUNGHI reduce cuvîntul eliminînd primul caracter pînă se obţine cuvîntul vid (fără nici un caracter).

Să se scrie o procedură asemănătoare care extrage ultimul caracter la fiecare execuţie.

Liste

Listele sînt structuri de obiecte grupate între paranteze drepte [şi]. Elementele unei liste sînt cuvinte separate prin blancuri sau alte liste delimitate de paranteze drepte.

EXEMPLE DE LISTE:

[BINE AŢI VENIT !] Lista formată din mai multe cuvinte.
[A1 +A2 =40] Lista formată din cuvinte, numere şi semne speciale.
[BINE] Lista ce conţine un cuvînt.
[ ] Lista vidă, nu conţine nici un element.
[ACEASTĂ LISTĂ CONŢINE [O SUB LISTĂ] CA ELEMENT]
[ACEASTĂ CONŢINE [O SUB [SUB LISTĂ]]]

O listă vidă notată cu [ ] este asemănătoare cu un cuvînt vid notat cu " urmat de blanc dar nu este acelaşi lucru.

Acesta este de fapt un caz special al regulii generale că în Logo niciodată o listă nu este acelaşi lucru cu un cuvînt. De exemplu [CUVÎNT] şi "CUVÎNT

sînt structuri de date diferite chiar dacă

PR "CUVÎNT și PR [CUVÎNT]
CUVÎNT *CUVÎNT*

se execută la fel. Acesta este un alt caz particular al aceleiași legi generale referitoare la Cuvinte și Liste. Listele sînt utilizate pentru a crea structuri de date complexe în general, nu numai pentru a grupa cuvinte în propoziții și fraze.

Procedura TRIUNGHI.CUV poate fi rescrisă foarte ușor pentru a prelucra liste într-o manieră asemănătoare.

EXEMPLU:
```
TO TRIUNGHI.LIST :LISTA
IF LISTA=[ ] THEN STOP
TRIUNGHI.LST BUTFIRST :LISTA
END
TRIUNGHI.LIST [ACEASTA ESTE O LISTĂ NU UN CUVÎNT]
ACEASTA ESTE O LISTĂ NU UN CUVÎNT
ESTE O LISTĂ NU UN CUVÎNT
O LISTĂ NU UN CUVÎNT
LISTA NU UN CUVÎNT
NU UN CUVÎNT
UN CUVÎNT
CUVÎNT
```

Un lucru foarte important care trebuie reținut este că atît numerele, cît și cuvintele și listele pot fi transferate ca intrări și ieșiri ale procedurilor.

Delimitatori

Cel mai utilizat delimitator este spațiul (blancul) care separă la ambele capete un cuvînt de restul liniei. Uneori însă se poate renunța la spații de delimitare dacă cuvintele sînt separate prin operatori sau alte semne speciale. Caracterele care pot fi considerate delimitatori sînt:

[ ] ( ) = > < + − * / Astfel:
1>2 + (3+4) / 5 − 6 este corect
1 > 2 + (3 + 4) / 5 − 6 este corect și mai clar

Trebuie remarcat modul specific în care Logo utilizează parantezele. În primul rînd trebuie bine făcută distincția între paranteze drepte [ ] și paranteze rotunde ( ). Parantezele drepte delimitează o listă.

EXEMPLU:
[ROȘU ALB] — o listă cu două elemente, Roșu și Alb;
[3+4 5] — tot o listă cu două elemente, 3+4 și 5;
[CULOARE] — o listă cu un element, cuvîntul CULOARE;
[ ] — lista vidă, fără nici un element.

Parantezele rotunde arată o grupare care nu este întotdeauna asemănătoare cu utilizarea în matematică.

EXEMPLU:
```
PRINT 3+4 * 5 poate fi scris cu paranteze
23
PRINT (3+4) * 5 ca în matematică
35
```

Dar expresia $x^2 + y^2$ se poate scrie astfel:
OUTPUT SUM (SQRT :X) (SQRT :Y)

Comanda OUTPUT are nevoie de o intrare care este suma celor două intrări ale operaţiei SUM care sînt operaţiile SQRT : X şi SQRT : Y. De remarcat că (SQRT : X) arată gruparea operaţiei cu intrările sale şi nu este acelaşi lucru cu SQRT (: X) care de fapt este incorect. Deci în Logo parantezele rotunde arată o grupare şi nu sînt semne speciale care delimitează lista intrărilor unei funcţii ca în BASIC.

Pentru operaţiile aritmetice SUM, PRODUCT este necesară utilizarea parantezelor dacă operaţia are mai mult de două intrări:

EXEMPLU:

```
PRINT    SUM   3   4   5 nu este corect
(PRINT   SUM   3   4   5) este corect
12
```

Variabile şi atribuirea numerelor în Logo

În exemplele precedente se poate constata că procedurile şi variabilele de intrare au nume care facilitează referirea la aceste entităţi. O variabilă poate fi privită ca un container pentru un obiect. Obiectul respectiv este *valoarea* care se atribuie variabilei. Variabila la rîndul ei este referită printr-un nume. O variabilă poate fi creată prin asocierea unor intrări nedefinite numelui unei proceduri sau prin utilizarea primitivei MAKE. În Logo se utilizează comanda MAKE ca o modalitate generală de a atribui lucrurilor un nume.

EXEMPLU:

```
MAKE   "NUMĂR 5        MAKE   "NUMĂR 10
PRINT  :NUMĂR          PRINT  :NUMĂR
5                      10
```

Primul parametru pentru comanda MAKE este numele care va fi atribuit obiectului care este transmis prin al doilea parametru.

De aici înainte referirile la acest obiect se pot face prin numele atribuit. Comanda MAKE poate fi utilizată în diferite moduri exemplificate mai jos.

EXEMPLE:

```
MAKE     "CULOARE    "ALBASTRU
PRINT    "CULOARE
CULOARE

PRINT    :CULOARE
ALBASTRU

MAKE     "FRAZA      [CERUL ESTE SENIN]
PRINT    :FRAZA
CERUL ESTE SENIN

PRINT SENTENCE (BUTLAST :FRAZA) :CULOARE
CERUL ESTE ALBASTRU

MAKE     "FLOARE     "COLORATA
PRINT    :FLOARE
COLORATĂ

MAKE     :FLOARE     [CU ROŞU ŞI CU ALB]
PRINT    :FLOARE
COLORATĂ
```

```
PRINT      :COLORATĂ
CU ROŞU ŞI CU ALB
```

Se constată din acest exemplu că numele asociat cu [CU ROŞU ŞI CU ALB] nu este literalul "FLOARE ci lucrul asociat acestuia, care este cuvîntul CULOARE.

EXEMPLU:

```
MAKE    :FLOARE [CU ROŞU ŞI CU ALB] au acelaşi efect ca şi
MAKE    "COLORATĂ [CU ROŞU ŞI CU ALB].
```

Pentru a vedea dacă unui obiect i s-a atribuit o valoare, adică dacă unui nume i s-a atribuit un lucru se utilizează comanda NAMEP care reîntoarce valoarea TRUE în caz afirmativ şi FALSE în caz negativ.

EXEMPLE:

```
PRINT      NAMEP      "ANOTIMP
FALSE

MAKE       "ANOTIMP   "VARA
PRINT      :ANOTIMP
VARA

PRINT      NAMEP      "ANOTIMP
TRUE
```

Mai mult decît atît, funcţia THING reîntoarce după apelare lucrul asociat intrării sale, sau altfel spus reîntoarce valoarea asociată unui nume. Astfel THING "X este acelaşi lucru cu : X care poate fi considerată o prescurtare. Dar dacă THING : X este corect din punct de vedere sintactic : : X nu este.

EXEMPLE:

```
MAKE       "NUME [MORARU AL. DAN]
PRINT      :NUME
MORARU AL. DAN

PRINT THING "NUME
MORARU AL. DAN

PRINT THING (WORD "NU "ME)
MORARU AL. DAN

PRINT THING (FIRST [NUME ADRESA])
MORARU AL, DAN

TO  INC    :X
MAKE       :X  1  +  THING  :X
END

MAKE       "TOTAL 7
PRINT      :TOTAL
7

INC        "TOTAL
PRINT      :TOTAL
8

INC        "TOTAL
PRINT      :TOTAL
9
```

Variabile locale şi globale

Regulile de lucru cu variabile în Logo sînt următoarele:

O variabilă declarată ca intrare într-o procedură, deci apare în linia cu titlul definiţiei, este locală acestei proceduri şi subprocedurilor apelate. După terminarea execuţiei procedurii variabilele locale nu mai au nici o valoare. Astfel, mai multe proceduri pot utiliza variabile locale cu acelaşi nume fără să apară vreun conflict. Acest lucru este valabil şi dacă în cadrul procedurii se utilizează comanda MAKE pentru a modifica valorile variabilelor de intrare, după cum se arată în exemplele următoare:

EXEMPLE:

```
TO      SCRIE    :X
PRINT            :X
INCR             :X
PRINT            :X
END

TO      INCR     :X
MAKE "X          :X +1
PRINT            :X
END

SCRIE   1
1       — tipărit de SCRIE
2       — tipărit de INCR
1       — tipărit de SCRIE
```

Trebuie observat că deşi INCR modifică valoarea lui X la 2, a doua afişare a lui X în SCRIE este tot 1, deoarece INCR a modificat valoarea lui X proprie procedurii SCRIE.

O variabilă creată în mod Logo, deci la nivel de comandă cu primitiva MAKE este o variabilă globală.

Astfel de variabilă nu este asociată vreunei proceduri.

Numele variabilelor globale nu interferează în nici un fel cu numele variabilelor locale. În exemplele următoare se ilustrează aceste reguli.

EXEMPLE:

```
MAKE    "X 1
INCR    :X
2
PRINT   :X
1

MAKE    "X 3
SCRIE   :X
3
4
3
```

O variabilă creată cu comanda MAKE într-o procedură, dar care nu apare ca intrare în linia de definire este o variabilă globală. Astfel de variabile se numesc în Logo *variabile libere*.

Pentru determinarea valorii asociate unui nume, în prezenţa variabilelor libere, se face în felul următor:

— Dacă numele apare printre variabilele de intrare ale procedurii, valoarea face parte din setul de valori proprii procedurii;

— Dacă nu, se verifică dacă numele este în setul propriu procedurii care a apelat procedura curentă;

— Dacă nu, se verifică dacă numele este în setul propriu procedurii care a apelat acea procedură, şi aşa mai departe pînă se ajunge la setul de valori globale definite în mod comandă.

Variabilele libere constituie un mecanism puternic de comunicaţie între proceduri. Utilizarea fără precauţii deosebite poate duce însă la programe greu de înţeles şi chiar la erori nedepanabile mai ales dacă se utilizează comanda MAKE pentru schimbarea valorilor variabilelor libere.

EXEMPLU:

```
TO      NIV1
MAKE    "Y 15
NIV2    :Y
END

TO      NIV2
PRINT   :Y
END

NIV1
15
```

Variabila Y este creată în procedura NIVI înainte de utilizarea în procedura NIV2. Deci Y este global. (variabilă liberă).

Cu următoarele definiţii se prezintă un alt exemplu:

EXEMPLU:

```
TO      NIV1
NIV1    15
PRINT   :Y
END

TO      NIV2     :Y
END

NIV1
Y has no value in NIV 1.
```

Acest mesaj arată că Y este locală subprocedurii NIV2 şi nu este accesibilă superprocedurii NIV1.

10.7. Reguli privind lucrul cu proceduri

Facilităţile de definire şi utilizare a procedurilor constituie o caracteristică importantă, care conferă putere de expresie deosebită limbajului Logo. Astfel, o problemă complexă poate fi rezolvată cu paşi mici, mai uşor de controlat de către programator. În Logo există proceduri predefinite denumite proceduri *primitive* şi proceduri definite de utilizator. În capitolul următor sînt prezentate împreună cu exemple de utilizare toate procedurile primitive grupate după

funcţiile lor. În acest paragraf se vor analiza cîteva aspecte comune privind definirea şi utilizarea procedurilor.

Procedurile definite de programator sînt formate din proceduri primitive.

EXEMPLU:

```
HIDETURTLE
SHOWTURTLE
```

Sînt proceduri primitive care fac ca penelul să dispară de pe ecran, respectiv să apară pe ecran.

EXEMPLU:

```
TO    SCRIE
PRINT [BINE AŢI VENIT LA CERCUL DE LOGO]
END
SCRIE defined
```

Aceasta este o procedură definită de programator care scrie mesajul dintre parantezele drepte ori de cîte ori este apelată, utilizînd primitiva PRINT. Definiţia unei proceduri începe totdeauna cu linia de definire a titlului care începe cu TO apoi numele procedurii şi eventualii parametri de intrare. Ultima linie este totdeauna END. Deoarece o procedură întotdeauna face ceva, s-a ales TO ca şablon de început de definire a unei proceduri. TO SCRIE este ca şi cum am zice A SCRIE, TO DRAW este A DESENA, etc.

Apelarea procedurii se face prin introducerea de la tastatură a numelui acesteia.

SCRIE şi calculatorul afişează pe ecran
BINE AŢI VENIT LA CERCUL DE LOGO.

Să definim o procedură care o utilizează pe SCRIE.

EXEMPLU:

```
TO    SALUT
SCRIE
PRINT [CRED CA O SĂ VĂ PLACĂ]
PRINT "SUCCES
END

SALUT
BINE AŢI VENIT LA CERCUL DE LOGO
CRED CĂ O SĂ VĂ PLACĂ
SUCCES
```

SCRIE este o subprocedură pentru SALUT iar SALUT este o superprocedură pentru SCRIE. Dacă se apelează o procedură care nu a fost definită şi nu este o procedură primitivă se emite un mesaj de eroare.

JUMP
I don't know how to JUMP.

Proceduri cu intrări

Procedurile SCRIE şi SALUT afişează întotdeauna acelaşi lucru. Ele se execută la fel pentru fiecare apelare. PRINT este apelată de trei ori şi de fiecare

dată scrie pe ecran ceea ce primeşte ca parametru la intrare. Acestea sînt intrări specificate explicit.

În definirea unor proceduri complexe intrările pot fi şi implicite, date de ieşirile altor proceduri.

Să definim o procedură care desenează un pătrat pe ecran utilizînd comenzile de deplasare a penelului.

EXEMPLU:

```
TO     PĂTRAT
REPEAT 4 [FORWARD 20 RIGHT 90]
END
```

La fiecare apel se va desena un pătrat cu latura 20 de puncte pe ecran. (Dacă nu pare a fi pătrat aceasta se datorează distanţei diferite pe orizontală şi pe verticală dintre două puncte de pe ecran).

Pentru a desena un pătrat cu orice latură se defineşte:

EXEMPLU:

```
TO     PATRATV      :LATURA
       REPEAT       4 [FORWARD :LATURA RIGHT 90]
END
```

Acum latura pătratului este specificată la apelare PĂTRATV 20 va avea acelaşi efect ca prima definiţie PĂTRAT. Trebuie notat că la apelare se scrie PĂTRATV 20, PĂTRATV 100 şi nu PĂTRATV : 20 sau PĂTRATV :100. La definire prin :LATURA am înţeles valoarea laturii. La apelare se specifică chiar valoarea, deci nu mai trebuie să apară : ca la definire.

Pentru a desena un dreptunghi oarecare se defineşte:

EXEMPLU:

```
TO     DREPTUNGHI :LUNGIME  :LĂŢIME
       FORWARD :LĂŢIME
       RIGHT 90
       FORWARD: :LUNGIME
       RIGHT 90
       FORWARD :LĂŢIME
       RIGHT 90
       FORWARD :LUNGIME
       RIGHT 90
END
```

Procedura are două intrări şi s-a presupus că orientarea penelului este de jos în sus în momentul apelului. După execuţia procedurii penelul rămîne cu aceeaşi orientare.

DREPTUNGHI 50 50 va desena un pătrat. Reamintim faptul că numele care apar ca intrări ale procedurilor (deci apar în linia cu titlul definiţiei) sînt variabile private şi nu interferează cu alte proceduri. Deci mai multe proceduri pot avea acelaşi nume ca variabile de intrare fără să apară conflict.

Tipuri de proceduri

Am văzut că după modul în care sînt definite, procedurile sînt de două tipuri: primitive şi definite de programator. După modul în care se execută, procedurile sînt iarăşi de două tipuri: *comenzi* care nu generează o valoare de

ieşire şi efectul lor se vede imediat şi *operaţii* care generează la ieşire valori care pot fi utilizate mai departe ca intrări pentru alte comenzi sau operaţii. Aceasta înseamnă că pentru a avea sens o procedură de tipul operaţiei întotdeauna trebuie urmată de o altă procedură care face ceva cu ieşirile generate de operaţie. Mai jos se arată cîteva exemple:

EXEMPLU:

```
PRINT 2 * 3 +11
17
MAKE "A   12
```

Acestea sînt comenzi. Efectul lor este imediat. Afişează pe ecran 17, respectiv modifică valoarea variabilei A.

EXEMPLU:

```
TO     INCR :X
            :X+1
END
INCR   5
You don 't say what to do with 6 in INCR
```

Se generează acest mesaj de eroare pentru că valoarea de ieşire nu apare ca intrare într-o procedură.

EXEMPLU:

```
TO        INCR :X
OUTPUT         :X+1
END

PRINT     INCR 5
6
```

Aceasta este corectă. Valoarea :X + 1 este o intrare pentru primitiva OUTPUT. Ieşirea generată de INCR este o intrare pentru comanda PRINT. Decarece o comandă nu generează nimic la ieşire ea nu poate fi utilizată ca intrare pentru o altă comandă.

EXEMPLU:

```
PRINT        FORWARD 50
```

După ce penelul se deplasează înainte cu 50 paşi se emite mesajul:
FORWARD does not output to PRINT.

Procedurile definite de utilizator sînt fie operaţii, fie comenzi, la fel ca şi procedurile primitive.

Introducerea şi editarea procedurilor

Pentru introducerea şi editarea procedurilor trebuie să se ţină seama de modul de lucru în care Logo este la un moment dat.

Mod Logo (nivelul cel mai de sus). Caracterul ? apare la începutul fiecărei linii Logo. Acesta este modul *direct* de lucru în care fiecare instrucţiune introdusă de utilizator este interpretată şi executată imediat, de aceea se mai numeşte şi *mod comandă*.

Mod TO (de scriere a unei proceduri). În acest mod se intră din mod comandă dacă TO este primul cuvînt dintr-o linie de program Logo.

La începutul fiecărei linii de ecran apare caracterul > pe durata în care Logo este în mod TO.

Modul editare (EDITOR). În acest mod se pot crea proceduri noi sau se pot modifica proceduri definite în prealabil. Intrarea în mod editor se face cu comanda EDIT sau ED urmată de numele procedurii şi a variabilelor de intrare. În colţul din stînga sus a ecranului apare şi dispare alternativ caracterul.

În paragraful Logo pe HC 85 se prezintă utilizarea tastaturii în regim de editare.

Ecranul poate fi în unul din următoarele două moduri:

— *Text* în care sînt disponibile 22 linii cu defilare implicită la umplerea ecranului.

— *Grafic* în care primele 22 de linii sînt la dispoziţia utilizatorului pentru grafice şi texte iar ultimele două linii sînt utilizate pentru scrierea de texte (ca un mic ecran în mod TEXT).

Ieşirea din mod comandă, deci ieşirea din Logo se poate face cu comanda BYE. Revenirea în LOGO se face cu RUN. Zona de lucru a rămas intactă şi se poate continua.

10.8. Expresii condiţionale

Într-o procedură comenzile şi operaţiile sînt interpretate şi executate de Logo linie cu linie, în ordinea în care sînt scrise de programator. Dacă se întîlneşte numele unei proceduri se întrerupe execuţia curentă pentru a se executa acea subprocedură după care se revine pentru a continua execuţia comenzii sau operaţiei următoare apelului subprocedurii. Deşi utilizarea procedurilor este principalul instrument de structurare a unui program complex, există şi alte modalităţi de modificare a secvenţei în care Logo citeşte şi execută instrucţiunile.

Instrucţiunea IF şi predicate

Forma generală a instrucţiunii IF pentru implementarea pe HC 85 a limbajului Logo este:

IF predicat lista instrucţiuni 1 lista instrucţiuni 2

Prima intrare pentru IF este un predicat, o operaţie logică sau altfel spus condiţia pe care o testează IF.

Predicatele sînt operaţii care generează TRUE sau FALSE după evaluarea expresiei logice de test. Dacă rezultatul este TRUE se execută lista instrucţiuni 1 iar dacă este FALSE se execută lista instrucţiuni 2, dacă este prezentă.

Dacă lista instrucţiuni 2 lipseşte, execuţia continuă cu instrucţiunea următoare.

EXEMPLE:

```
TO   SEMN
     IF :N <0 [OUTPUT "NEGATIV] [OUTPUT "POZITIV]
END
```

```
PRINT    SEMN 17
POZITIV
PRINT    SEMN (5—7)
NEGATIV
TO       ALEGE
IF       0=RANDOM 3 [OUTPUT "DA]
OUTPUT   "NU
END
PRINT    ALEGE
DA
```

În aceste exemple IF este o comandă care generează la ieşire POZITIV sau NEGATIV, DA sau NU.

În exemplul următor IF este utilizată ca o operaţie:

EXEMPLU:

```
TO       ALEGEO
         OUTPUT IF 0=RANDOM 3 ["DA] ["NU]
END
PRINT    ALEGEO
DA
```

Predicate predefinite

Pentru operaţiile cu numere au fost introduse predicatele < > şi =. Există şi alte predicate predefinite ca funcţii primitive în Logo. Acestea pot fi uşor reperate deoarece, de obicei, numele acestora se termină cu P. În alte implementări numele predicatelor predefinite sau definite de programator se termină cu ?.

EXEMPLE DE PREDICATE:

SHOWNP	care generează la ieşire TRUE dacă penelul este în starea activat, afişat pe ecran, altfel generează FALSE.
EMPTYP	obiect care generează TRUE dacă obiectul este fără conţinut, altfel FALSE.
EQUALP	obiect 1 obiect 2 forma prefixată a operaţiei =. Generează TRUE dacă obiect 1 şi obiect 2 sînt egale, altfel generează FALSE.
LISTP	obiect. Generează la ieşire TRUE dacă obiect este o listă, altfel generează FALSE.
MEMBERP	obiect lista. Generează TRUE dacă obiect aparţine listei, altfel FALSE
NUMBERP	obiect. Generează TRUE dacă obiect este număr, altfel FALSE.
WORDP	obiect Generează TRUE dacă obiect este cuvînt, altfel FALSE.
NAMEP	obiect Generează TRUE dacă obiect are o valoare, altfel FALSE.
KEYP	Generează TRUE dacă s-a apăsat pe o tastă sau pe o combinaţie de taste recunoscute de Logo, altfel FALSE.
DEFINEDP	cuvînt Generează TRUE dacă cuvînt este numele unei proceduri, altfel FALSE.
PRIMITIVEP	cuvînt Generează TRUE dacă cuvînt este numele unei primitive Logo, altfel FALSE.

Predicate definite de utilizator

Pe lîngă predicatele predefinite se pot defini noi predicate deoarece predicatele nu sînt altceva decît proceduri care generează la ieşire cuvintele TRUE sau FALSE.

EXEMPLE:

```
POZITIVP       :X Generează la ieşire TRUE dacă x ≥ 0, altfel FALSE
TO POZITIVP :X
```

```
IF X <0          [OUTPUT "FALSE] [OUTPUT "TRUE]
END
VOCALAP     :C generează TRUE dacă cuvîntul C începe cu o vocală, astfel FALSE
TO VOCALAP :C
IF (FIRST :C) = "A          [OUTPUT TRUE]
IF (FIRST :C) = "E          [OUTPUT TRUE]
IF (FIRST :C) = "I          [OUTPUT TRUE]
IF (FIRST :C) = "O          [OUTPUT TRUE]
IF (FIRST :C) = "U          [OUTPUT TRUE]
END
```

După ce un predicat a fost definit de către programator, el poate fi utilizat în instrucţiunea IF ca şi predicatele predefinite.

Pentru a facilita combinarea predicatelor în Logo există primitivele AND, NOT şi OR care efectuează operaţii logice cu predicate şi care generează rezultate tot de tipul TRUE sau FALSE.

Astfel, pentru definirea unui predicat care determină dacă un număr este cuprins între două limite se poate proceda în diferite moduri.

```
EXEMPLE:
TO INTREP      :X      :MIN   :MAX
IF :X < :MIN                 [OUTPUT FALSE]
IF :X > :MAX                 [OUTPUT FALSE]
OUTPUT TRUE
END
TO INTREP :X        :MIN     :MAX
IF OR (:X < :MIN)   (:X> :MAX) [OUTPUT FALSE]
OUTPUT TRUE
END
TO INTREP   :X     :MIN    :MAX
IF AND (NOT (:X < :MIN)) (NOT (:X >  :MAX)) [OUTPUT TRUE]
OUTPUT FALSE
END
TO INTREP   :X     :MIN    :MAX
OUTPUT AND (NOT (:X < :MIN)) (NOT (:X > :MAX))
END
TO INTREP   :X     :MIN    :MAX
OUTPUT NOT OR (:X < :MIN) (:X > : MAX)
END
```

Deoarece AND şi OR sînt la rîndul lor predicate care generează TRUE sau FALSE în ultimele două exemple aceste valori sînt transmise direct procedurii care cheamă pe INTREP.

10.9. Linii complexe şi interpretarea lor în Logo

În acest paragraf se vor analiza unele aspecte privind scrierea şi interpretarea unor linii complexe în Logo.

O linie de program Logo poate avea pînă la 242 de caractere, deci mult mai lungă decît o linie de pe ecran. O linie se termină efectiv cînd se introduce de la tastatură ENTER. La introducerea unei linii de program complexe, la sfîrşitul liniei de ecran se trece implicit la linia de ecran următoare iar semnul ! care este scris de Logo arată că linia de ecran următoare este continuarea aceleiaşi linii de program Logo.

La prima vedere notaţia prefixată pune unele prob eme privind „citirea" unei linii complexe în Logo.

O linie de program este în general interpretată ca o secvenţă de cuvinte separate prin blancuri sau alţi delimitatori.

Să considerăm o linie de program Logo care generează produsul six (x) * cos (x)

EXEMPLU:

FD 50 comandă mişcarea penelului cu 50 de paşi înainte,
FD50 este o eroare şi Logo emite mesajul:
I don "t know how to FD50.

exceptînd cazul în care a fost definită în prealabil procedura cu numele FD50.

OUTPUT PRODUCT (SIN :X) (COS :X)

Parantezele arată o grupare a unei operaţii cu intrarea sa.

Această linie se poate scrie şi fără paranteze astfel:

OUTPUT PRODUCT SIN :X COS :X şi se citeşte în felul următor:

Primul cuvînt întîlnit este OUTPUT care are nevoie de o intrare. Căutînd această intrare continuă explorarea liniei de la stînga la dreapta şi următorul cuvînt găsit este PRODUCT. Dar PRODUCT este o operaţie care necesită două intrări, deci Logo continuă cu căutarea acestora. Următorul cuvînt găsit este SIN care este o funcţiune ce necesită o intrare.

Apoi Logo găseşte variabilă :X pe care o atribuie ca intrare pentru SIN şi astfel s-a lămurit şi prima intrare pentru PRODUCT, explorarea liniei continuă şi Logo găseşte funcţia COS, apoi intrarea :X şi cu aceasta s-a lămurit şi cea de-a doua intrare pentru PRODUCT. Rezultatul operaţiei PRODUCT constituie intrarea pentru OUTPUT.

Se poate trage concluzia că atîta timp cît se ştie cîte intrări are fiecare procedură şi dacă toate procedurile sînt operatori prefixaţi pentru intrări nu este nevoie de paranteze, iar explorarea unei linii se face generalizînd exemplul de mai sus. Pe de altă parte, utilizarea parantezelor pentru grupări uşurează atît scrierea cît şi interpretarea liniei complexe de program.

Pentru operatori infixaţi trebuie notat că operatorii aritmetici infixaţi, * /, + — au prioritate mai mare decît operatorii prefixaţi.

EXEMPLU:

WORD 5+3 9	**dă rezultatul 89 deoarece se execută**
5+3=8	**apoi funcţia WORD. Aceeaşi expresie se poate scrie mai explicit:**
WORD (5+3) 9	**Pentru a asigna valoarea —17 variabilei cu numele X se scrie:**
MAKE "X (—17)	**expresie corectă, sau**
MAKE "X —17	**semnalată ca eroare deoarece**

Logo încearcă să scadă 17 din "X.

În sfîrşit, operatorii infixaţi >, <, = au prioritate mai mică decît operatorii prefixaţi.

Pentru procedurile cu un număr variabil de intrări, trebuie utilizate paranteze pentru grupare dacă numărul intrărilor este mai mare decît cel implicit (vezi SUM, PRODUCT, etc).

Deci regulile pentru interpretarea unei linii Logo sînt:

La întîlnirea numelui unei proceduri trebuie ştiut dacă:

— este o comandă sau o operaţie

— cîte intrări trebuie să aibă.

Prima procedură dintr-o linie Logo trebuie să fie întotdeauna o comandă.

O operaţie apare ca intrare pentru o altă procedură.

Trebuie determinată fiecare intrare a unei proceduri.

La completarea intrărilor pentru o comandă, următoarea procedură trebuie să fie tot o comandă.

10.10 Glosar de termeni Logo

— **adresă**: locaţia unui registru sau a unei celule de memorie prin care se specifică sursa sau destinaţia datelor.

— **apel**: activarea (lansarea în execuţie) a unui program, a unei proceduri sau subproceduri.

— **ASCII**: American Standard Code for Information Interchange; cod standard de reprezentare în calculatoare a informaţiilor alfanumerice. Fiecărei cifre sau semn special inclus în standard i se asociază în mod unic un cod format din 7 biţi. Ex: A=1000001, B=1000010, 3=0110011 etc.

— **binar**: ceva cu două stări stabile posibile; de ex: sistemul binar de numeraţie se bazează pe cifrele 0 şi 1.

— **bit**: o cifră binară; unitate elementară de reprezentare a informaţiilor în calculatoare sub forma unor vectori de biţi.

— **boot**: (bootstrapare): procesul de încălcare în memoria calculatorului a unui program de sistem sau aplicativ.

— **buffer** (tampon): o zonă de memorie utilizată ca zonă tampon pentru păstrarea temporară a datelor în cazul transferurilor de la un echipament la altul. O zonă de memorie unde se citesc datele sau de unde se scriu datele în operaţiile de intrare/ieşire.

— **bug**: (pană): o eroare într-un pogram.

— **byte**: vezi octet.

— **call**: vezi apel.

— **caracter**: o literă, o cifră sau alt simbol utilizat pentru organizarea, controlul şi reprezentarea datelor.

— **colectare spaţiu disponibil** (garbage collection): recuperarea spaţiului de memorie care a fost utilizat, dar nu mai este, în vederea reutilizării lui.

— **comandă**: o procedură Logo, fie că este primitivă sau este definită de programator, care face ceva într-un mod particular, fără să genereze ceva la ieşire spre procedura care a chemat-o. Ex. CLEAR SCREEN, FORWARD, PRINT sînt exemple de comenzi.

— **comanda calculatorului**: Dacă în maniera convenţională calculatoarele erau folosite în învăţămînt pentru a programa elevii — adică pentru a comanda comportarea lor — filozofia încapsulată în Logo se bazează pe ideea că elevii să programeze calculatoarele, să le controleze în mod direct. Se spune adesea că cea mai bună cale de a învăţa este predîndu-le altora. Calculatorul este un elev foarte receptiv şi devotat profesorului său. Controlul comportării calculatorului poate constitui şi o bună experienţă de conducere pentru elev.

— **condiţional**: o linie Logo care are ca efect executarea unor instrucţiuni diferite, funcţie de o condiţie testată.

— **coordonată**: un număr care descrie poziţia pe orizontală sau verticală a penelului pe ecranul grafic al calculatorului. În poziţia iniţială penelul are coordonatele [0, 0].

— **cursor**: un marcaj deplasabil utilizat pentru a indica o poziţie pe ecran.

— **cuvînt vid**: un cuvînt care nu are nici un caracter. Un cuvînt vid se scrie astfel

— **defilare**: deplasarea imaginii pe ecran, sau a unei părţi a acesteia, pe verticală sau orizontală pentru a face loc unor date să fie afişate în porţiunea utilă a ecranului iar datele de pe marginea cealaltă a ecranului dispar din cîmpul vizual.

— **depanare**: găsirea şi eliminarea greşelilor dintr-un program, îmbunătăţind astfel comportarea unui program care nu face ceea ce se doreşte de la el. În Logo se consideră ca programele care nu se comportă exact cum s-a dorit sînt programe neterminate, nu terminate prost. În general în şcoală compoziţiile scrise ale elevilor sînt notate aşa cum sînt, nu după ce au fost depanate. Aceasta din motive obiective, inclusiv efortul prohibitiv care ar fi necesar dacă elevii ar fi încurajaţi să-şi îmbunătă-

ţească lucrările scrise. În Logo elevii sînt încurajaţi să studieze cu atenţie penele din programele lor. Există mari şanse ca o comportare nedorită a unui progam, dacă este bine studiată să poată duce la idei noi chiar mai interesante decît ideea iniţială. Dintr-o pană, se poate învăţa întotdeauna ceva.

— **dispozitiv**: un echipament ataşat calculatorului, de exemplu o imprimantă, un terminal grafic.

— **editare**: introducerea (de la tastatură), modificarea sau ştergerea unui text reprezentînd proceduri, date sau orice şiruri de caractere.

— **editor**: acea porţiune din Logo care asistă programatorul şi-l ajută să editeze un text alfanumeric într-o zonă de memorie numită zona de editare.

— **element**: un membru dintr-o mulţime sau un termen dintr-o serie.

— **execută**: comandă Logo să efectueze o instrucţiune sau o procedură. Aceasta se poate face furnizînd numele procedurii direct de la tastatură sau indirect, ca instrucţiune într-o subprocedură.

— **exersare penel**: programatorul se substituie penelului şi parcurge o procedură grafică aşa cum este ea văzută de penel. Aceasta poate duce la o bună înţelegere a unor construcţii geometrice complexe, fără o pregătire corespunzătoare în geometrie. Pe un plan mai larg, acest proces trebuie privit mai mult ca o parcurgere mintală a unei proceduri înainte de programarea ei, decît ca o parcurgere a unei proceduri existente. De exemplu, desenarea unui cerc cu penelul, ştiind că singurele comenzi sînt deplasarea înainte şi rotirea, poate fi uşor înţeleasă prin exersarea penelului.

— **fenomenul QWERTY**: Papert utilizează acest fenomen pentru a se referi la situaţii păstrate prin tradiţie care deşi au avut iniţial o motivaţie corectă, au devenit învechite, demodate. Numele vine de la cele 6 litere din partea stîngă sus a unei tastaturi de calculator. Aranjarea tastelor s-a făcut iniţial pentru a evita suprapunerea mai multor taste prin „încetinirea" vitezei de lucru a operatorului (taste care sînt frecvent alăturate în text au fost aranjate la distanţă pe tastatură). Mai multe încercări de a optimiza aranjarea tastelor au eşuat datorită obişnuinţei de a lucra cu vechea aranjare. Astfel multe din programele şcolare supravieţuiesc datorită obişnuinţei, chiar dacă nu mai sînt adecvate.

— **figură deschisă**: o figură, o formă formată dintr-o linie care nu se mai întoarce în punctul de start.

— **figură închisă**: o figură, o formă formată dintr-o linie continuă care se reîntoarce în punctul iniţial.

— **fişier**: o colecţie de informaţii organizată şi memorată pe un suport în vederea unor prelucrări ulterioare.

— **fişier AŞCII**: un fişier de tip text format din şiruri de caractere reprezentate în cod ASCII.

— **format**: o aranjare a datelor, conform cu anumite şabloane, pe ecran, imprimată, bandă sau disc magnetic.

— **global**: care se aplică la întreg domeniul considerat. Variabilele globale sînt accesibile şi pot fi modificate din orice parte a programului. Utilizarea variabilelor locale — a căror domeniu de valabilitate este o procedură — este de dorit deoarece conduce la programe mai ordonate, mai clare şi mai uşor de depanat. Sistemul de coordonate cartezian permite specificarea globală a punctelor unei figuri, faţă de origine. Aceasta creează dificultăţi importante în reprezentările grafice, fapt pentru care Logo este orientat în general spre reprezentări locale, relative.

— **grafica cu penelul**: efectuarea unor exerciţii de grafică cu penelul are un mare succes atît pentru introducerea în programare pentru orice vîrstă, pentru însuşirea unor concepte fundamentale din programare cît şi ca bază pentru o programă de matematică asistată de calculator.

— **heading**: vezi orientare.

— **imagini antropomorfice**: metafore prin care calculatoare, proceduri sau obiecte controlate de calculatoare sînt privite ca şi cum ar fi persoane. Imaginile antropomorfice facilitează transferul de cunoştinţe dintr-un mediu familiar spre un nou context. Gîndindu-ne la maşini ca la nişte oameni (limitaţi) sau chiar modelînd comportarea oamenilor prin analogii cu procese ce se desfăşoară în diferite maşini nu implică tratarea oamenilor ca pe maşini. Prin învăţarea penelului să se mişte după legile lui Newton se poate spune că penelul a descoperit o nouă lume. Aceasta nu este decît o metaforă pentru activitatea de programarea calculatorului.

— **implicit**: o valoare sau o opţiune furnizată de program dacă utilizatorul nu specifică nimic atunci cînd el poate sau trebuie să specifice aceste valori sau opţiuni.

— **instruire asistată de calculator (CAI)**: în sens larg, orice activitate educațională care utilizează calculatoare. Logo presupune utilizarea calculatorului nu numai pentru furnizarea, exersarea și testarea cunoștințelor ci mai degrabă ca un context de utilizare și dezvoltare a gîndirii în scopul rezolvării problemelor de interes specific. Instruirea programată modelează omul ca o mașină în sensul că elevul este programat de calculator. Utilizarea limbajului Logo consideră elevul ca un factor constructiv care construiește cunoștințe în mod activ. Elevul ia rolul profesorului învățînd penelul o nouă lume.

— **inteligența artificială**: o ramură a științei calculatoarelor din care a izvorît și Logo. Numele acestei ramuri a științei derivă din încercările de dezvoltare a unor metode și tehnici de utilizare a mașinilor pentru a simula comportamentul considerat inteligent al oamenilor și animalelor. Această privire îngustă nu mai este valabilă. Psihologia cunoașterii și inteligența artificială sînt deseori referite împreună ca știința cunoașterii. Printre domeniile de activitate ale IA sînt: înțelegerea limbajului natural, percepți a vizuală, acumularea și reprezentarea cunoștințelor.

— **interactiv**: un mod de lucru pe bază de dialog între calculator și utilizator.

— **intrare/ieșire**: intrarea unei proceduri reprezintă un mesaj necesar procedurii pentru a-și executa funcția. Unele proceduri au nevoie de mai multe mesaje de intrare pentru a ști ce trebuie să facă. De exemplu o procedură de desenare a unui poligon oarecare are nevoie de două intrări, numărul de pași pe care-i face înainte (lungimea laturii) și unghiul de întoarcere la fiecare colț. Ieșirea unei proceduri reprezintă mesajul transmis de o procedură spre procedura care a apelat-o, procedură care produce o ieșire (o operație) trimite mesajul de ieșire spre o procedură care-l așteaptă ca intrare. De exemplu mesajul transmis de procedura SUM este așteptat de PRINT pentru a-l tipări. Intrările și ieșirile sînt strict corelate în Logo.

— **iterație**: execuție repetitivă, de către calculator, a unei secvențe de instrucțiuni. În Logo se preferă recursivitatea pentru implementarea structurilor de control repetitive.

— **întreg**: un număr pozitiv sau negativ care nu conține o parte fracționară.

— **linie de continuare**: o linie de program Logo care reprezintă extensia unei linii de ecran precedente. Punctul de întrerupere a liniei este marcat cu semnul exclamării!.

— **LISP**: prescurtarea pentru List Processing. Un limbaj de programare foarte utilizat în cercetările din domeniul inteligenței artificiale, sursa multor idei din Logo.

— **listă**: structura de date de bază în Logo. Lista este o secvență ordonată de obiecte arbitrare. Deoarece un obiect poate fi un cuvînt, un număr sau altă listă (care la rîndul ei poate conține alte liste, și asta pe oricîte nivele), listele pot fi utilizate pentru a crea structuri de date foarte complexe.

— **lista vidă**: o listă care nu are nici un element. O listă vidă se scrie astfel [ ].

— **local**: ceva care are sens (se aplică) într-un context (domeniu) specific în numai în cadrul procedurii în care a fost definit. Într-o altă procedură poate fi definită o altă variabilă locală cu același nume fără ca ele să interfere. Modul de deplasare a penelului este de asemenea local, fără să fie nevoie să se țină seama de poziția absolută în raport cu o originefixă. Pentru a desena un cerc, penelul ia în considerare doar porțiunea din plan din jurul poziției curente. Spre deosebire de aceasta, generarea cercului după execuția $x^2+y^2=r^2$ are un caracter global, într-un sistem de coordonate global.

— **Logo**: este o prescurtare ce derivă din cuvîntul grecesc pentru „cuvînt" sau „gînd".

— **notație infixată**: un mod de a scrie expresiile aritmetice astfel că operatorul este plasat între operanzi.

— **notație prefixată**: un mod de scriere a expresiilor aritmetice astfel că operatorul este plasat înaintea operanzilor.

— **număr real**: un număr fracționar pozitiv sau negativ.

— **obiect**: entitate specifică de reprezentare a datelor în Logo. Logo operează cu obiecte reprezentînd cuvinte și liste. Deși numerele sînt tot cuvinte, uneori sînt privite ca obiecte separate. Unele implementări includ obiecte de tip tablou. Obiectele sînt prelucrate, prin intermediul lor se transmit mesaje de intrare și ieșire în cadrul procedurilor. Deși procedurile sînt privite ca elemente active, ele pot fi manipulate (create, distruse, modificate, transmise înainte sau înapoi)? de către alte proceduri. Deci și procedurile pot fi uneori privite ca obiecte.

— **octet**: un vector binar format din 8 biţi. Lungimea cuvintelor cu care operează majoritatea calculatoarelor este de 1 sau mai mulţi octeţi.

— **operaţie**: o procedură în Logo care calculează o valoare şi o transmite procedurii chemătoare. Valoarea respectivă poate fi un număr, un cuvînt sau o listă.

— **operaţie logică**: un predicat care are ca intrări valorile TRUE sau FALSE.

— **orientare**: direcţia, în grade, spre care este orientat penelul cu vîrful colorat al triunghiului. Este un număr cu valoarea între 0 şi 359 şi este furnizat la ieşirea operaţiei HEADING; 0 corespunde direcţiei Nord, 90 — Est, 180 — Sud şi 270 — Vest.

— **pană**: vezi bug.

— **penel**: peniţă electronică ale cărei mişcări pe ecranul de afişare sînt controlate de către calculator prin comenzi de mişcare (FORWARD, BACK) sau comenzi de rotaţie (LEFT, RIGHT). Pe ecran penelul apare ca un triunghi mic cu un vîrf colorat care denotă şi orientarea, direcţia pe care se va deplasa la o comandă. Prin deplasare, penelul lasă urma deplasării sale pe ecran şi poate fi folosit în acest mod pentru a desena pe ecran. În limba engleză se utilizează termenul de „turtle".

— **peneldin**: penel dinamic. Aşa cum penelul (static) are poziţia spaţială şi orientarea fixate prin comenzi Logo, penelul dinamic poate avea viteza sau acceleraţia fixate. O comandă de schimbare a stării modifică coordonatele spaţiale sau orientarea penelului static sau modifică viteza penelului dinamic, avînd un efect analog aplicării unei forţe. Peneldin se comportă ca o rachetă în spaţiul extraterestru. Pentru a deplasa nava este necesară aplicarea unui impuls prin aprinderea unei rachete. Aceasta se mişcă apoi în aceeaşi direcţie pînă primeşte un nou impuls. Noua mişcare este o combinaţie între mişcarea veche şi mişcarea cauzată de impulsul aplicat. Penelul dinamic este un instrument deosebit în studierea unor probleme de fizică şi matematică.

— **Piaget**: Jean Piaget, considerat unul dintre marii gînditori în domeniul educaţiei. Observînd comportamentul copiilor în fiecare, atunci cînd invaţă, are un rol activ în propria-i dezvoltare. A arătat valoarea examinării şi înţelegerii unui răspuns greşit. Un răspuns greşit nu trebuie privit ca o mişcare aleatoare în lipsa unui răspuns corect, ci ca o pană într-un program care în cea mai mare parte oferă răspunsul corect. Trebuie examinată „logica gîndirii" care a dus la acel răspuns sau explicaţie. Aceasta este modul în care în Logo sînt privite penele dintr-un program şi depanarea acestora.

— **pixel**: un punct grafic pe ecran.

— **predicat**: o procedură care generează la ieşire TRUE sau FALSE.

— **primitive**: procedurile intrinseci ale limbajului Logo. Acestea pot fi utilizate de către programatori pentru construirea propriilor proceduri.

— **procedură**: element conceptual de bază în construirea programelor în Logo. Programatorul are iniţial un vocabular format din proceduri primitive pe care le poate utiliza pentru definirea de noi proceduri care sînt utilizate exact în aceeaşi manieră ca şi cele primitive.

— **prompt**: un semn afişat de către calculator pe ecran pentru a atenţiona utilizatorul să furnizeze informaţii.

— **program**: datorită utilizării extensive a procedurilor, putem spune că procedura de pe cel mai înalt nivel — o superprocedura execută un program de obicei foarte simplu sau care constă numai din apelul mai multor subproceduri.

— **propoziţie**: o listă de cuvinte. În Logo sînt furnizate unelte puternice de prelucrare a cuvintelor unei propoziţii.

— **recursivitatea**: o procedură sau o funcţie recursivă este definită în termenii ei însăşi. Prin recursivitate o problemă foarte complicată poate fi exprimată în termenii unor versiuni mai simple ale aceleaşi probleme. În general o definire recursivă a unei proceduri este mai eficientă decît o definire repetitivă, iterativă. De multe ori recursivitatea este singura cale de rezolvare a unei probleme.

— **rezolvare de probleme**: deprindere care constituie punctul central al oricărei activităţi în Logo. Cea mai mare parte a instruirii tradiţionale în matematică (şi alte discipline) se referă la probleme rezolvate, la însuşirea tehnicilor altora de rezolvare a problemelor. Logo încearcă implementarea ideii de rezolvare a unor probleme noi. Ca tehnică se utilizează divizarea unei probleme complexe în părţi mai mici prin utilizarea extensivă a procedurilor.

— **scroll**: vezi defilare.

— **sintaxă**: regurile care stau la baza structurii unui limbaj.

— **spaţiu de lucru**: o parte a spaţiului de memorie a calculatorului utilizată pentru a memora variabile, proceduri şi proprietăţi într-o sesiune de lucru în Logo.

— **stare**: o parte din istoria evoluţiei unui lucru care permite ca pe baza unor acţiuni asupra lui să putem determina evoluţia ulterioară. Prin stare se exprimă proprietăţile relevante la un moment dat. De exemplu, starea penelului la un moment dat este dată de poziţia şi direcţia în care este orientat. Cunoscînd starea curentă, aceasta este suficient de relevantă deoarece prin aplicarea unor comenzi prin program se poate determina comportarea penelului. Posibilitatea de izolare, de decupare a aspectelor importante (relevante) ale unei situaţii este un instrument de depanare foarte eficient.

— **stivă**: o structură de date astfel că ultimul element memorat în stivă va fi primul element extras în vederea prelucrării.

— **subprocedură**: o procedură care este apelată de o altă procedură. O procedură poate fi subprocedura pentru ea însăşi, vezi recursivitate.

— **şir**: o secvenţă de caractere.

— **ştiinţa de carte în calculatoare**: în general prin ştiinţa de carte se înţelege posibilitatea de a scrie şi a citi. În mod similar ştiinţa de carte în calculatoare a fost văzută ca o experienţă generală şi adeseori superficială, în calculatoare. Cunoaşterea principiilor de funcţionare şi de organizare a unui calculator şi posibilitatea de exprimare fluentă într-un limbaj de programare constituie sensul dat în această carte termenului respectiv. Oferind elevilor posibilitatea de a fi programatori sînt ajutaţi în exprimarea fluentă a ideilor matematice şi logice. Aceasta poate apărea mai degrabă ca învăţarea unei limbi străine — o opţiune foarte bună pentru cei ce s-au hotărît să o facă, dar nu neapărat necesară pentru a trăi şi a munci. Răspîndirea foarte largă a calculatoarelor în toate domeniile de activitate oferă un statut nou, obiectiv, ştiinţei de carte în calculatoare.

— **tampon**: vezi buffer.

— **valoare**: conţinutul unei variabile. Valoarea este asociată unei variabile cu un anumit nume.

— **variabilă**: un container (locaţie de memorie) care are un nume şi care conţine o valoare.

— **workspace**: vezi spaţiu de lucru.

Capitolul 11. Primitivele limbajului LOGO, cu exemplificări

11.1. Caracteristici generale ale primitivelor LOGO

LOGO este un limbaj flexibil, de tip procedural. Limbajul dispune de proceduri implicite denumite *primitive*. Aceste primitive stau la baza definirii procedurilor utilizator, cu ajutorul cărora se descriu algoritmii care implementează diverse programe. Din punctul de vedere al programatorului nu există deosebire între primitivele limbajului şi procedurile construite de el în ceea ce priveşte folosirea acestora.

În acest capitol se vor prezenta primitivele limbajului LOGO cu ajutorul cărora utilizatorii îşi vor descrie procedurile şi programele lor.

Principalele grupuri de primitive sînt:

— primitive pentru controlul penelului şi ecranului în mod grafic şi alfanumeric;

— primitive pentru specificarea operaţiilor matematice şi logice;

— primitive pentru manipularea obiectelor LOGO (variabile, numere, cuvinte, liste);

— primitive pentru comunicaţia cu echipamentele de intrare/ieşire;

— primitive care asigură ramificaţiile în program;

— primitive pentru lucrul cu fişiere şi gestiunea spaţiului de lucru;

— primitive pentru editarea procedurilor;

— primitive pentru diferite funcţii.

În limbajul LOGO exista două tipuri de primitive:

— comenzi şi

— operaţii.

Comenzile sînt acele primitive care au o acţiune directă în momentul apelării, de exemplu: diferite operaţii asupra penelului, încărcarea/salvarea procedurilor, listarea/ ştergerea procedurilor, etc. fără a produce o valoare care să fie utilizată de o altă primitivă sau procedură.

Operaţiile sînt acele primitive care generează o valoare care va fi utilizată de o altă primitivă sau procedură.

În continuare se vor descrie diversele tipuri de primitive ale limbajului LOGO sub forma:

— denumirea primitivei;

— mnemonica primitivei şi intrările asociate acesteia;

— mnemonica prescurtată şi intrările asociate;
— descrierea funcţiei efectuată de primitivă;
— exemplu de utilizare a primitivei.

11.2. Primitive LOGO pentru controlul penelului şi al ecranului în regim grafic

În limbajul LOGO ecranul calculatorului poate fi utilizat în două regimuri de lucru:
a. mod grafic = ecranul este considerat o suprafaţă cu 256 puncte pe orizontal şi 175 puncte pe vertical;
b. mod alfanumeric = ecranul este considerat o suprafaţă cu 22 de linii a cîte 32 caractere fiecare.

În regim grafic există un cursor grafic numit **penel** (pensula cu care se pictează sau prescurtare de la **peniţa** electronică), ce poate fi controlat de utilizator prin intermediul primitivelor LOGO. În limba engleză, pentru acest cursor grafic, se foloseşte noţiunea „turtle" (tradus: „broscuţa").

În limbajul LOGO există o serie de primitive care oferă utilizatorului posibilitatea de a controla ceea ce se vede pe ecran în regim grafic şi de a afla (specifica) starea ecranului şi a penelului.

Primitivele vor fi grupate în patru categorii:
— primitive pentru schimbarea stării penelului;
— primitive care specifică starea penelului;
— primitive pentru utilizarea penelului şi a ecranului;
— primitive care specifică starea penelului şi a ecranului.

Pentru mişcarea penelului pe ecran se pot folosi comenzi de:
— mişcare absolută (faţă de originea axelor ecranului);
— mişcare relativă (faţă de ultima poziţie a penelului pe ecran)—prin specificarea distanţei de deplasare şi a unghiului de rotire a direcţiei.

Avînd în vedere că mişcarea penelului se face prin specificarea distanţei de deplasare şi a direcţiei este necesar să precizăm axele ecranului şi sensul direcţiei.

În limbajul LOGO ecranul în regim grafic, arată ca în figura 11.1

La iniţierea unei sesiuni de lucru în limbajul LOGO, penelul este considerat în originea axelor, punctul de coordonate 0, 0 şi orientat spre direcţia Nord.

Orice schimbare a direcţiei penelului în sensul mişcării acelor de ceas se consideră spre dreapta, iar în sensul opus spre stînga.

Primitivele din cadrul fiecărei categorii se vor descrie, în general, în ordine alfabetică pentru a uşura accesul utilizatorului.

11.2.1. Primitive pentru schimbarea stării penelului

În acest paragraf se vor descrie o serie de comenzi care vor controla mişcarea penelului, pe ecranul aflat în regim grafic.

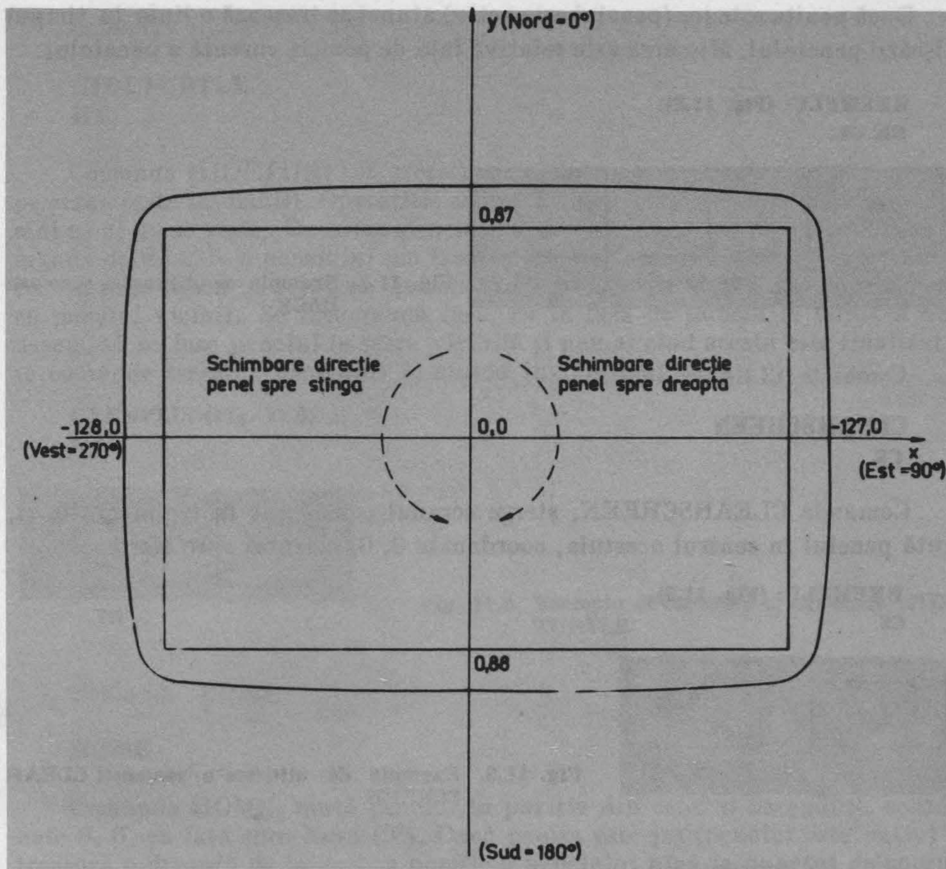


Fig. 11.1. Orientarea şi limitele poziţionale ale ecranului în regim grafic

Comenzile care vor fi tratate în acest paragraf sînt:

BACK	RIGHT
CLEARSCREEN	SETHEADING
FORWARD	SETPOS
HIDETURTLE	SETX
HOME	SETY
LEFT	SHOWTURTLE

Comanda BACK

BACK d

BK d

Comanda BACK, mută penelul înapoi (în sensul invers vîrfului penelului) cu distanţa „d" paşi (puncte), specificată în cadrul comenzii. Direcţia penelului nu se schimbă. De notat faptul că dacă distanţa „d" este negativă, mişcarea se face înainte:

(BK −d ≡ FD d)

Dacă peniţa este jos (penelul este activ) atunci se trasează o linie în timpul mişcării penelului. Mişcarea este relativă faţă de poziţia curentă a penelului.

EXEMPLU: (Fig. 11.2).
BK 50

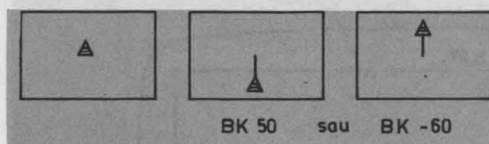


Fig. 11.2. Exemplu de utilizare a comenzii BACK

Comanda CLEARSCREEN

CLEARSCREEN
CS

Comanda CLEARSCREEN, şterge ecranul considerat în regim grafic şi, mută penelul în centrul acestuia, coordonate 0, 0, orientat spre Nord.

EXEMPLU: (Fig. 11.3).
CS

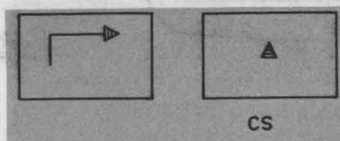


Fig. 11.3. Exemplu de utilizare a comenzii CLEAR SCREEN

Comanda FORWARD

FORWARD d
FD d

Comanda FORWARD, mută penelul înainte (în sensul indicat de vîrful penelului) pe distanţa „d" paşi. Direcţia penelului nu se schimbă. De notat faptul că dacă distanţa „d" este negativă mişcarea se face înapoi. (FD —d ≡ BK d). Dacă peniţa este jos (penelul este activ), atunci se trasează o linie în timpul mişcării penelului. Mişcarea este relativă faţă de poziţia curentă a penelului.

EXEMPLU: (Fig. 11.4).
FD 50

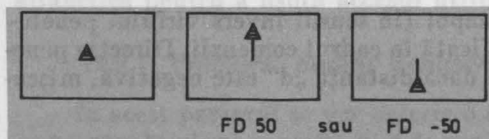


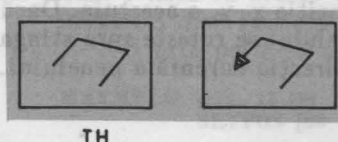
Fig. 11.4. Exemplu de utilizare a comenzii FORWARD

Comanda HIDETURTLE

HIDETURTLE
HT

Comanda HIDETURTLE, trece penelul într-o stare în care nu se mai vede pe ecran (este invizibil). Operaţiile asupra sa au acelaşi efect ca pînă acum numai că el nu se vede. Deoarece penelul nu se mai desenează pe ecran, orice comandă de mişcare a penelului (cu trasare sau nu) se execută mai repede. Efectuarea unui desen pe ecran cu penelul invizibil se realizează mai repede decît cu penelul vizibil. Se recomandă însă, ca în faza de punere la punct a unui desen, să se lase penelul în stare vizibilă şi numai cînd acesta este finalizat să se comande trecerea penelului în starea invizibilă.

EXEMPLU: (Fig. 11.5)
HT



TH

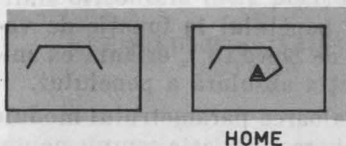
Fig. 11.5. Exemplu de utilizare a comenzii HIDETURTLE

Comanda HOME

HOME

Comanda HOME, mută penelul în poziţia din centrul ecranului, coordonate 0, 0, cu faţa spre Nord (0°). Dacă peniţa este jos (penelul este activ) se trasează o dreaptă de la vechea poziţie a penelului pînă la punctul de coordonate 0,0.

EXEMPLU: (Fig. 11.6)
HOME



HOME

Fig. 11.6. Exemplu de utilizare a comenzii HOME

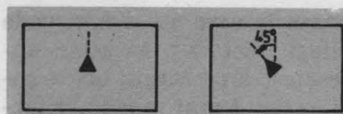
Comanda LEFT

LEFT gr
LT gr

Comanda LEFT roteşte direcţia penelului cu „gr" grade spre stînga (invers mişcării acelor de ceas) fără a schimba poziţia x, y, a acestuia. Dacă parametrul „gr" are valoare negativă direcţia penelului se roteşte spre dreapta ((LT −gr ≡ RT gr). Rotirea este relativă faţă de direcţia curentă a penelului.

EXEMPLU: (Fig. 11.7).

LT 45



LT 45

Fig. 11.7. Exemplu de utilizare a comenzii LEFT

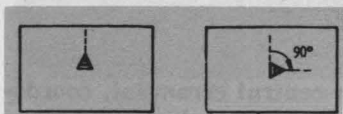
Comanda RIGHT

RIGHT gr
RT gr

Comanda RIGHT roteşte direcţia penelului cu „gr" grade spre dreapta (în sensul mişcării acelor de ceas), fără a schimba poziţia x, y, a acestuia. Dacă parametrul „gr" are valoarea negativă direcţia penelului se roteşte spre stînga (RT —gr ≡ LT gr). Rotirea este relativă faţă de direcţia curentă a penelului.

EXEMPLU: (Fig. 11.8).

RT 90



RT 90

Fig. 11.8. Exemplu de utilizare a comenzii RIGHT

Comanda SETHEADING

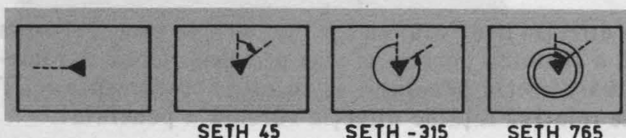
SETHEADING gr
SETH gr

Comanda SETHEADING, stabileşte direcţia penelului în funcţie de valoarea parametrului „gr", exprimat în grade, faţă de Nord (0°), orientarea iniţială a ecranului. Parametrul „gr" specifică direcţia absolută a penelului.

Direcţia penelului se stabileşte considerînd valoarea parametrului modulo 360. Dacă „gr" are valoarea negativă atunci direcţia se stabileşte complementar faţă de 360°.

EXEMPLU: (Fig. 11.9).

SETH 45 ≡ SETH —315 ≡ SETH 765



SETH 45 SETH -315 SETH 765

Fig. 11.9. Exemplu de utilizare a comenzii SETHEADING

Comanda SETPOS

SETPOS [x y]

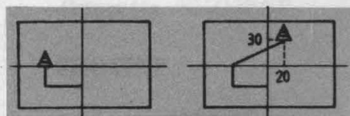
Comanda SETPOS mută penelul în poziţia determinată de coordonatele absolute x y, specificate sub formă de listă [x y]. Dacă penelul este activ se trasează o dreaptă între poziţia curentă a cursorului şi poziţia specificată de coordonatele x y. Direcţia penelului nu se schimbă. Coordonatele x y pot depăşi valorile maxime admise pentru ecran, dacă acesta este configurat cu comanda WINDOW, dar în acest caz el nu se va vedea pe ecran. Dacă coordonatele depăşesc limitele prezentate în figura 11.1, se indică eroare în cazul în care ecranul a fost configurat cu comanda FENCE. În cazul în care ecranul a fost configurat cu comanda WRAP, valorile se iau modulo limite.

Trebuie menţionat faptul că dacă dorim să poziţionăm penelul într-un punct de coordonate :X :Y — reprezentate de valorile unor variabile, va trebui să creăm o listă cu aceste variabile, utilizînd operaţia SENTENCE:

SETPOS SENTENCE :X :Y

— NU este permisă construcţia SETPOS [:X :Y]

EXEMPLU: (Fig. 11.10)
SETPOS [20 30]



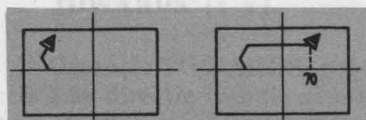
SETPOS [20 30] **Fig. 11.10. Exemplu de utilizare a comenzii STPOS**

Comanda SETX

SETX x

Comanda SETX, muta penelul orizontal în punctul de abscisă x, ordonata y rămînînd nemodificată. Dacă penelul este activ (peniţa jos), se va trasa o linie orizontală între poziţia curentă şi poziţia specificată.

EXEMPLU: (Fig. 11.11)
SETX 70



SETX 70 **Fig. 11.11. Exemplu de utilizare a comenzii SETX**

Comanda SETY

SETY y

Comanda SETY, mută penelul în punctul de ordonată y, abscisa x rămînînd nemodificată. Dacă penelul este activ (peniţa jos), se va trasa o linie ver-

ticală între poziţia curentă şi poziţia specificată.

EXEMPLU: (Fig. 11.12).
SETY 75

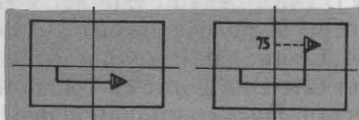


Fig. 11.12. Exemplu de utilizare a comenzii SETY

Comanda SHOWTURTLE

SHOWTURTLE
ST

Comanda SHOWTURTLE, trece penelul în starea în care se vede pe ecran (penelul vizibil). Are un efect invers faţă de comanda HIDETURTLE.

EXEMPLU: (Fig. 11.13)
ST

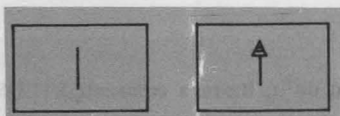


Fig. 11.13. Exemplu de utilizare a comenzii SHOWTURTLE

11.2.2. Primitive care specifică starea penelului

În acest paragraf se descriu operaţiile prin care utilizatorul poate afla informaţiile privind starea penelului.

Primitivele ce vor fi tratate în acest paragraf sînt:

HEADING	TOWARDS
POSITION	XCOR
SHOWNP	YCOR

Operaţia HEADING

HEADING

Operaţia HEADING, generează un număr natural între 0 şi 359, care specifică direcţia absolută a penelului. Numărul generat de operaţia HEADING poate fi utilizat ca parametru pentru diverse comenzi LOGO.

La începutul unei sesiuni de lucru LOGO direcţia penelului este spre Nord (0°).

EXEMPLU:

```
CS
LT 90
PRINT HEADING
270
```

Operaţia POSITION

POSITION
POS

Operaţia POSITION, generează coordonatele absolute ale poziţiei curente a penelului, sub forma unei liste [x y]. La începutul unei sesiuni de lucru apelul operaţiei POSITION va genera lista [0 0].

EXEMPLU:

```
FD 30
RT 90
FD 70
PRINT POSITION
70 30
```

Operaţia SHOWNP

SHOWNP

Operaţia SHOWNP generează valoarea logică TRUE, dacă penelul este în starea vizibilă şi valoarea logică FALSE, în caz contrar.

EXEMPLU:

```
ST
PRINT SHOWNP
TRUE
HT
PRINT SWOWNP
FALSE
```

Operaţia TOWARDS

TOWARDS [x y]

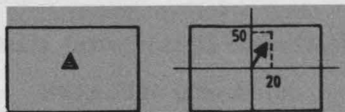
Operaţia TOWARDS, generează un număr natural între 0 şi 359, care specifică ce direcţie trebuie să aibă penelul pentru a fi îndreptat spre punctul de coordonate x, y. Coordonatele x, y, se specifică sub forma unei liste [x y].

Trebuie menţionat faptul că dacă dorim să îndreptăm penelul spre un punct ale cărui coordonate rezultă din valorile unor variabile :X :Y, va trebui să creăm o listă cu aceste variabile, utilizînd operaţia SENTENCE, astfel:

TOWARDS SENTENCE :X :Y

EXEMPLU: (Fig. 11.14)

```
SETHEADING TOWARDS [20 50]
```



SETHEADING TOWARDS [20 50] Fig. 11.14. Exemplu de utilizare a operației
FD 30 TOWARDS

Operația XCOR

XCOR

Operația XCOR generează un număr care specifică valoarea abscisei x corespunzătoare poziției curente a penelului.

EXEMPLU:

```
CS
RT 90
FD 40
PRINT XCOR
40
```

Operația YCOR

YCOR

Operația YCOR generează un număr care specifică valoarea ordonatei y corespunzătoare poziției curente a penelului.

EXEMPLU:

```
CS
FD 20
PRINT YCOR
20
```

11.2.3. Primitive pentru utilizarea peniței electronice și a ecranului

În acest paragraf se descriu comenzile care permit utilizatorului să controleze direct penelul și ecranul.

Primitivele care vor fi tratate în acest paragraf sînt:

CLEAN	SETBG
DOT	SETBORDER
FENCE	SETPC
PENDOWN	SETSCRUNCH
PENERASE	WINDOW
PENREVERSE	WRAP
PENUP	

Comanda CLEAN

CLEAN

Comanda CLEAN șterge ecranul, în regim grafic, fără să afecteze poziția și direcția penelului.

EXEMPLU: (Fig. 11.15)
CLEAN

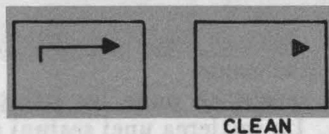


Fig. 11.15. Exemplu de utilizare a comenzii CLEAN

Comanda DOT

DOT [x y]

Comanda DOT, desenează un punct în poziţia determinată de coordonatele x y, coordonate specificate sub forma de listă [x y], fără ca penelul să-şi schimbe poziţia şi direcţia.

Trebuie menţionat faptul că dacă dorim să desenăm un punct într-o poziţie determinată de valorile unor variabile :X :Y, trebuie să creăm o listă cu aceste variabile utilizînd funcţia SENTENCE:

DOT SENTENCE :X :Y

Dacă coordonatele depăşesc limitele ecranului atunci va apare un mesaj corespunzător.

EXEMPLU: (Fig. 11.16)
DOT [0 0]

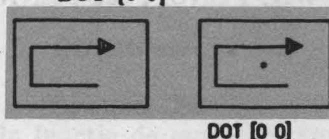


Fig. 11.16. Exemplu de utilizare a comenzii DOT

Comanda FENCE

FENCE

Comanda FENCE limitează mişcarea penelului în limitele ecranului (Fig. 11.1). După executarea acestei comenzi, orice încercare de mutare a penelului în afara limitelor ecranului produce un mesaj de eroare şi penelul nu îşi schimbă poziţia. Altfel spus, comanda FENCE nu permite trecerea penelului din zona vizibilă a ecranului în zona invizibilă (Fig. 11.21)

Dacă penelul se află deja în afara limitelor vizibile ale ecranului, comanda FENCE nu se validează.

EXEMPLU:
FENCE
CS
FD 200

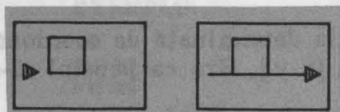
— se va genera mesajul de eroare:

Comanda PENDOWN

PENDOWN
PD

Comanda PENDOWN activează penelul, în sensul că pune jos peniţa. Orice mişcare a penelului va trasa urma pe ecran. La iniţierea unei sesiuni de lucru LOGO penelul este implicit activ (peniţa jos).

EXEMPLU: (Fig. 11.17)
PENDOWN
FD 120



PENDOWN
FD 120

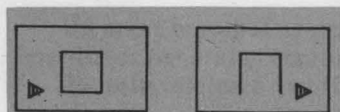
Fig. 11.17. Exemplu de utilizare a comenzii PENDOWN

Comanda PENERASE

PENERASE
PE

Comanda PENERASE va şterge punctele peste care trece penelul dacă anterior acestea au fost desenate. Efectul comenzii PENERASE este anulat prin utilizarea oricăreia dintre comenzile PENDOWN, PENUP, sau PENRE VERSE.

EXEMPLU: (Fig. 11.18)
PENERASE
FD 120



PENERASE
FD 120

Fig. 11.18. Exemplu de utilizare a comenzii PENERASE

Comanda PENREVERSE

PENREVERSE
PX

Comanda PENREVERSE va complementa punctele peste care trece penelul. Dacă acestea sînt desenate, atunci le şterge, iar dacă nu, atunci le desenează.

Efectul comenzii PENREVERSE este anulat prin utilizarea oricăreia dintre comenzile PENDOWN, PENUP sau PENERASE.

EXEMPLU: (Fig. 11.19)
PENREVERSE

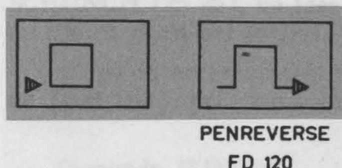


Fig. 11.19. Exemplu de utilizare a comenzii PENREVERSE

Comanda PENUP

PENUP
PU

Comanda PENUP face inactiv penelul, adică penița este ridicată și orice mișcare a sa nu va produce urmă pe ecran. Penelul nu va putea desena pînă cînd nu se activează acesta cu comanda PENDOWN.

EXEMPLU: (Fig. 11.20)
PENUP
FD 120

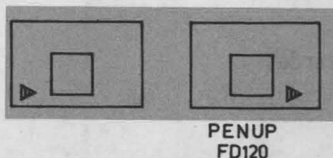


Fig. 11.20. Exemplu de utilizare a comenzii PENUP

Comanda SETBG

SETBG nrc

Comanda SETBG stabilește culoarea fondului ecranului în regim grafic, în funcție de numărul culorii „nrc" specificat. Corespondența între numărul culorii „nrc" și culoarea fondului este următoarea:

0 — negru	4 — verde
1 — albastru	5 — turcoaz
2 — roșu	6 — galben
3 — purpuriu (magenta)	7 — alb

EXEMPLU:
SETBG 6

Culoarea galben, devine culoarea fondului în regim grafic.

Comanda SETBORDER

SETBORDER nrc
SETBR nrc

Comanda SETBORDER stabilește culoarea chenarului în funcție de numărul culorii „nrc" specificat. Corespondența între numărul culorii „nrc" și culoarea chenarului este aceeași cu cea de la comanda SETBG.

EXEMPLU:
SETBORDER 1

Culoarea albastru devine culoarea chenarului.

Comanda SETPC

SETPC nrc

Comanda SETPC, stabileşte culoarea cu care penelul scrie pe ecran, în regim grafic, în funcţie de numărul culorii „nrc" specificat.

Corespondenţa între numărul culorii „nrc" şi culoarea cu care penelul scrie pe ecran este aceeaşi cu cea de la comanda SETBG.

Trebuie subliniat faptul că, la calculatorul HC-85, culoarea cu care se scrie (INK) poate fi stabilită numai la nivel de zonă caracter, nu la nivel de punct. De aceea o linie trasată cu o culoare nouă va schimba culoarea tuturor punctelor peste care trece şi a punctelor din zona caracter asociată. Din acest motiv trebuie avut grijă de modul în care se schimbă culorile pentru ca desenul să nu piardă din acurateţe.

EXEMPLU:
SETPC 4

Culoarea cu care scrie penelul va fi culoarea verde.

Comanda SETSCRUNCH

SETSCRUNCH [x y]

Comanda SETSCRUNCH stabileşte scala de reprezentare pe axele de coordonate, faţă de care se realizează un desen. La iniţierea unei sesiuni de lucru LOGO, implicit scara de reprezentare este [100 100]. În urma unei comenzi SETSCRUNCH această scală se poate modifica. Astfel:

SETSCRUNCH [100 25]

modifică o figură ce se va desena, pe ecranul calculatorului în sensul că lăţimea rămîne aceeaşi (scala pe x a rămas nemodificată), iar înălţimea este de 4 ori mai mică.

SETSCRUNCH [25 25]

va modifica figura, micşorînd de 4 ori atît înălţimea cît şi lăţimea.

Important de menţionat faptul că dacă dorim să stabilim scala pe axele de coordonate în funcţie de valorile unor variabile, va trebui să creăm o listă cu aceste variabile utilizînd funcţia SENTENCE, astfel:
SETSCRUNCH SENTENCE :X : Y

EXEMPLU:

```
REPEAT 4[FD 80 RT 90]
SETSCRUNCH [100 25]
REPEAT 4 [FD 80 RT 90]
SETSCRUNCH [25 25]
REPEAT 4[FD 80 RT 90]
```

Pătratul desenat în urma comenzii SETSCRUNCH [100 25] va deveni un dreptunghi la care înălţimea este un sfert din lăţime. Ultima comandă SET-

SCRUNCH [25 25], va face ca pătratul ce se va desena să aibă latura de 4 ori mai mică decît cel inițial.

După încercarea acestui exemplu este necesar să revenim la scala inițială de [100 100], printr-o comandă corespunzătoare.

Comanda WINDOW

WINDOW

Comanda WINDOW va permite ca penelul să poată fi poziționat în afara limitelor ecranului prezentate în Fig. 11.1. Putem face afirmația că ecranul are o dimensiune mare și numai o porțiune delimitată de limitele din Fig. 11.1 este vizibilă, Fig. 11.21.

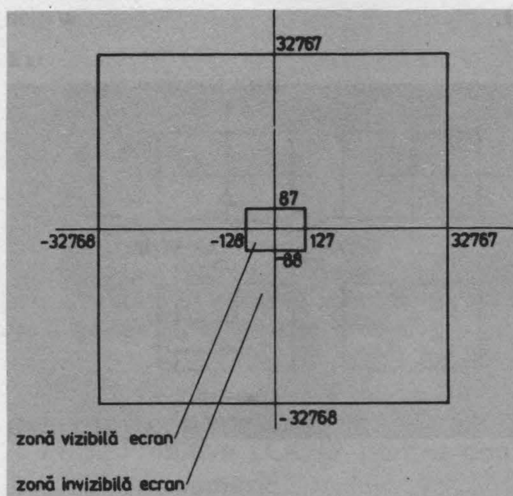


Fig. 11.21. Limitele zonei vizibile și zonei invizibile ale ecranului în cadrul comenzii WINDOW

Practic, comanda WINDOW permite penelului să treacă din zona vizibilă în zona invizibilă, fără ca limbajul LOGO să emită mesaj de eroare. Orice mișcare a penelului în afara zonei vizibile se va executa, numai că utilizatorul nu o va vedea pe ecran.

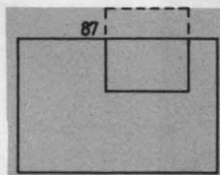
La inițierea unei sesiuni de lucru LOGO, implicit se consideră atît zona vizibilă cît și cea invizibilă (ca și cum s-ar fi specificat inițial o comandă WINDOW).

De notat faptul că, dacă penelul se află în zona invizibilă, limitarea mișcării penelului numai la zona vizibilă printr-o comandă FENCE va produce un mesaj de eroare.

EXEMPLU: (Fig. 11.22)

```
WINDOW
CS
REPEAT 4 [FD 120 RT 90]
```

120



CS
WINDOW
REPEAT 4[FD120 RT90]

Fig. 11.22. Exemplu de utilizare a comenzii WINDOW

Din pătratul cu latura de 120 se va vedea pe ecran numai zona continuă.

Utilizarea unei comenzi SETSCRUNCH [50 50], va modifica scala de reprezentare și deci figura se va micșora (latura pătratului se va înjumătăți) și se va putea vedea tot pătratul.

Comanda WRAP

WRAP

Comanda WRAP, face ca zona vizibilă a ecranului să fie continuă (marginea de sus a ecranului să fie adiacentă cu marginea de jos, iar marginile laterale să fie de asemenea adiacente (Fig 11.23). În acest caz, cînd penelul traversează o latură a ecranului va reapare imediat în latura opusă considerată adiacentă.

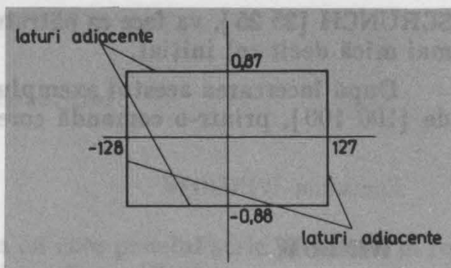


Fig. 11.23. Adiacenta laturilor ecranului în urma execuţiei comenzii WRAP

Punctul de coordonate (0, 88) este identic cu cel de coordonate (0, -88), iar punctul (0, 89) identic cu (0, -87).

EXEMPLU: (Fig. 11.24)

WRAP
REPEAT 4 [FD 120 RT 90]

Zona din pătrat, care în cadrul exemplului din figura 11.22. era invizibilă, a fost desenată jos, considerîndu-se laturile de sus şi jos ale ecranului ca fiind adiacente.

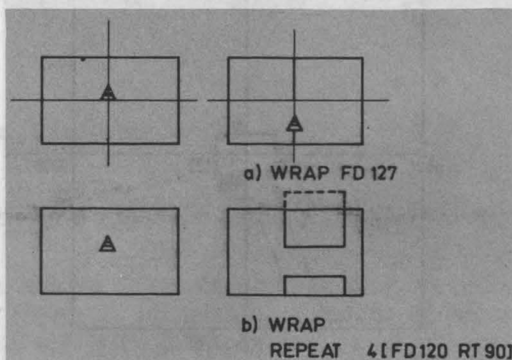


Fig. 11.24. Exemplu de utilizare a comenzii WRAP

11.2.4. Primitive care specifică starea peniţei şi a ecranului.

În acest paragraf se descriu operaţiile prin care utilizatorul poate afla informaţii privind starea peniţei şi a ecranului.

Primitivele care vor fi tratate în acest paragraf sînt:

BACKGROUND
PENCOLOUR
SCRUNCH

Operaţia BACKGROUND

BACKGROUND
BG

Operaţia BACKGROUND, generează un număr între 0 şi 7 ce reprezintă culoarea fondului ecranului. Corespondenţa între numărul generat şi culoarea fondului este aceeaşi cu cea descrisă la comanda SETBG.

La iniţierea unei sesiuni de lucru LOGO, culoarea fondului este alb. Este operaţia inversă a comenzii SETBG.

Ex:

PRINT BACKGROUND
7

Operaţia PENCOLOUR

PENCOLOUR
PC

Operaţia PENCOLOUR, generează un număr între 0 şi 7 ce reprezintă culoarea cu care scrie penelul. Corespondenţa între numărul generat şi culoarea cu care scrie penelul este aceeaşi cu cea descrisă la comanda SETBG.

La iniţierea unei sesiuni de lucru LOGO, culoarea cu care scrie penelul este negru.

Ex:

PRINT PENCOLOUR
0

Operaţia SCRUNCH

SCRUNCH

Operaţia SCRUNCH generează o listă formată din două numere care reprezintă scala pe axele de coordonate. La iniţierea unei sesiuni de lucru LOGO, scala de reprezentare este [100 100].

11.3. Primitive LOGO pentru controlul ecranului în regim alfanumeric

În regim alfanumeric, ecranul este considerat o suprafaţă organizată în 22 de linii a cîte 32 de caractere fiecare, (Fig. 11.25).

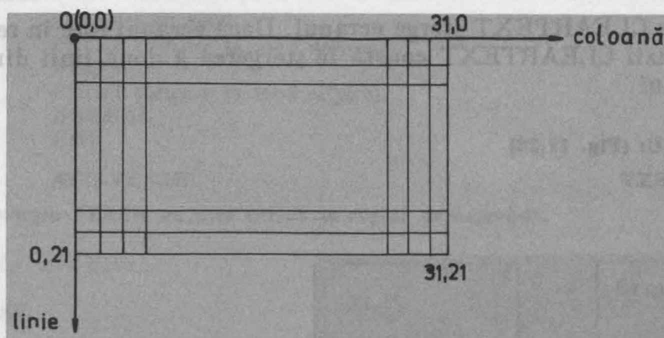


Fig. 11.25. Organizarea ecranului în mod alfanumeric

În regim de comandă în prima coloană este afişat prompterul ?, iar în regim de editare este afişat >.

Utilizatorul, cu ajutorul comenzilor LOGO poate stabili culoarea fondului ecranului şi culoarea caracterului.

Afişarea se poate realiza:

— în video normal (video direct) constă în scrierea cu culoarea caracterului pe un fond care are culoarea fondului;

— în video invers constă în scrierea cu culoarea fondului pe un fond care are culoarea caracterului. Altfel spus se interschimbă (inversează) culoarea caracterului cu cea a fondului.

Primitivele care vor fi tratate în acest paragraf sînt:

BRIGHT	NORMAL
CLEARTEXT	OVER
COPYSCREEN	SETCURSOR
CURSOR	SETTC
FLASH	TEXTSCREEN
INVERSE	

Comanda BRIGHT

BRIGHT b

Comanda BRIGHT, stabileşte nivelul de luminozitate a fondului ecranului în funcţie de valoarea parametrului „b", ce poate lua valori binare 1 sau 0.
Dacă: b = 1 — comanda stabileşte fond luminos;
b = 0 — comanda stabileşte fond normal.

EXEMPLU:

BRIGHT 1

Comanda CLEARTEXT

CLEARTEXT
CT

Comanda CLEARTEXT şterge ecranul. Dacă ecranul este în regim grafic, efectul comenzii CLEARTEXT constă în ştergerea a două linii din partea de jos a ecranului.

EXEMPLU: (Fig. 11.26)

CLEARTEXT

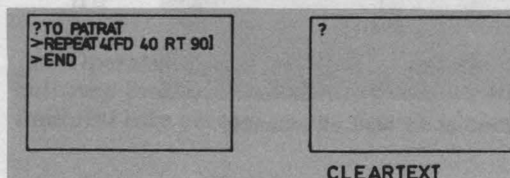


Fig. 11.26. Exemplu de utilizare a comenzii CLEARTEXT

Operaţia COPYSCREEN

COPYSCREEN

Operaţia COPYSCREEN copiază ecranul la imprimantă. Utilizatorul poate folosi această comandă indiferent de modul de organizare a ecranului alfanumeric sau grafic. Ea permite obţinerea unui „hard copy", la imprimantă.

Imprimanta trebuie să fie conectată în prealabil pe interfaţa paralelă corespunzătoare.

Observaţie: Această interfaţă constituie o dezvoltare ulterioară a calculatorului.

EXEMPLU:

COPYSCREEN

Operaţia CURSOR

CURSOR

Operaţia CURSOR generează o listă formată din două numere care specifică coordonatele *coloană*, *linie* a poziţiei curente a cursorului în regim alfanumeric (poziţia unde va apare primul caracter care va fi afişat pe ecran). Coordonatele coloană, linie vor fi în concordanţă cu cele prezentate în figura 11.25.

La iniţierea unei sesiuni de lucru LOGO, poziţia iniţială a cursorului este în partea stîngă sus a ecranului.

EXEMPLU:

PRINT CURSOR
0 1

Comanda FLASH

FLASH

Comanda FLASH stabileşte ca orice caracter ce se afişează pe ecran, după ce s-a executat această comandă, să apară în regim de clipire (se alternează, cu o anumită perioadă de timp, afişarea caracterului in video invers cu cea în video normal).

Ecranul rămîne în acest regim de afişare pînă cînd se dă o comandă NORMAL sau se produce o eroare.

EXEMPLU:

```
TO AFIŞ.FLASH
PRINT [afişare în mod normal]
FLASH
PRINT [afişare în mod clipire]
NORMAL
END

AFIŞ.FLASH
```

Observaţie: Funcţia FLASH nu este activă în regim de comandă.

Comanda INVERSE

INVERSE

Comanda INVERSE are ca efect interschimbarea culorii caracterului cu

cea a fondului. Se rămîne în acest regim de afişare pînă la o comandă NORMAL care trece în regim normal sau se produce o eroare.

EXEMPLU:

```
TO AFIŞ.INV
PRINT [afişare în mod normal]
INVERSE
PRINT [afişare în mod invers]
NORMAL
END
AFIŞ.INV
```

Comanda NORMAL

NORMAL

Comanda NORMAL are ca efect readucerea la modul normal de scriere (scriere cu culoarea caracterului pe culoarea fondului), indiferent de modul de afişare în care se află ecranul, INVERSE sau FLASH. Comanda NORMAL anulează efectul comenzilor INVERSE şi FLASH.

EXEMPLU:

```
TO AFIŞARE
PRINT [afişare în mod normal]
FLASH
PRINT [afişare în mod clipire]
NORMAL
INVERSE
PRINT [afişare în mod invers]
NORMAL
PRINT [afişare în mod normal]
END
AFIŞARE
```

Comanda OVER

OVER b

Comanda OVER stabileşte modul de trasare sau scriere în funcţie de parametrul „b" ce poate lua valori binare 1 sau 0.
Dacă:

b = 1 Se va scrie peste desenul sau textul de pe ecran în mod SAU EXCLUSIV (punctele care sînt stinse vor fi aprinse, iar cele care sînt aprinse vor fi stinse de trecerea penelului în regim grafic sau a cursorului în regim alfanumeric). Se va rămîne în această stare de scriere pînă cînd o comandă OVER 0 va fi executată sau se va produce o eroare;

b = 0 Se va scrie în regim normal.

EXEMPLU:

```
CLEARTEXT
TO AFIS.OVER
TS
SETCURSOR [10 1]
PRINT [textul se  şterge în regim OVER]
```

```
OVER 1
WAIT 25
SETCURSOR [10 1]
PRINT [textul se şterge în regim OVER]
OVER 0
END
AFIŞ.OVER
```

Comanda SETCURSOR

SETCURSOR [c l]

SETCUR [c l]

Comanda SETCURSOR, mută cursorul în regim alfanumeric, în poziţia determinată de coordonatele absolute c (coloana), l (linie) specificate sub formă de listă [c l]. Valorile parametrilor pot fi în următoarele limite:

— coloana „c" între 0 şi 31;

— linia „l" între 0 şi 21.

Depăşirea valorilor extreme ale parametrilor va produce un mesaj de eroare.

Pentru poziţionarea cursorului într-un punct de coordonate :X :Y reprezentate de valorile unor variabile va trebui să creăm o listă cu aceste variabile, utilizînd operaţia SENTENCE sau operaţia LIST.

EXEMPLU:

```
TEXTSCREEN
CLEARSCREEN
SETCURSOR [3 8]
PRINT [text din coloana 3 linia 8]
```

Comanda SETTC

SETTC [f c]

Comanda SETTC stabileşte culoarea fondului pe care se scrie şi culoarea cu care se scrie caracterul în funcţie de parametrii f (fond) şi c (caracter), specificaţi.

Parametrii f şi c sînt specificaţi sub formă de listă [f c] şi pot lua valori numerice între 0 şi 7.

Corespondenţa între numărul culorii fondului f, respectiv a caracterului c şi culorile utilizate este aceeaşi cu cea prezentată la comanda SETBG.

Pentru stabilirea culorilor fondului şi caracterului în funcţie de valorile unor variabile :F :C, va trebui să creăm o listă cu ele utilizînd operaţiile SENTENCE sau LIST.

EXEMPLU:

```
SETBORDER 1
SETTC [6 2]
```

Se stabileşte chenarul albastru şi se scrie cu culoarea roşie pe fond galben.

Comanda TEXTSCREEN

TEXTSCREEN
TS

Comanda TEXTSCREEN trece ecranul în regim alfanumeric.

În acest mod de lucru al ecranului nu se vede penelul şi nici desenele realizate cu el, ci numai textul procedurilor sau textul realizat prin execuţia programelor.

EXEMPLU:

TEXTSCREEN

11.4. Primitive LOGO pentru operaţii matematice şi logice

În acest capitol se prezintă operatorii şi primitivele care manipulează numere în vederea realizării unor operaţii matematice.

În limbaj LOGO se lucrează cu operanzi ce fac parte din:
— mulţimea numerelor întregi Z;
— mulţimea numerelor reale R.

Reprezentarea operanzilor din mulţimea R poate fi realizată în două moduri:

a) prin specificarea părţii întregi şi zecimale separate prin punct (.)

EXEMPLU: 2.793; —796.69189

Punctul (.) înlocuieşte virgula (,) pe care o utilizăm în scrierea numerelor zecimale la matematică.

b) prin specificarea mantisei şi a exponentului:

<mantisa> {E} <exponent>

EXEMPLU: 2.793162E4 ≡ 27931.62
8.7134E-3 ≡ 0.0087134
Valorile: 2.793162, respectiv 8.7134 reprezintă mantisa numărului iar E 4 reprezintă 10^4, respectiv E-3 reprezintă 10^{-3};

EXEMPLU: 3.791E2 ≡ 3.791 × 10^2 =379.1

Utilizarea reprezentării cu mantisă şi exponent este necesară în cazul în care avem de reprezentat valori foarte mari sau foarte mici.

Domeniul de reprezentare a valorilor numerelor este:
— în mulţimea numerelor întregi: $-99999999 < N < 99999999$
— în mulţimea numerelor reale: $-10^{38} < N < 10^{38}$, iar cea mai mică valoare de reprezentare este: 0.000001×10^{-38}.

În cadrul reprezentării cu mantisă şi exponent, mantisa se poate exprima prin maxim 8 cifre semnificative, iar exponentul este cuprins între —38 şi +38.

Regulile de evaluare a expresiilor sînt aceleaşi cu cele din matematică.

O precizare suplimentară care trebuie făcută este aceea că, pentru primitivele matematice delimitarea unei expresii de parametri asupra căreia se aplică operatorul se realizează cu sau fără paranteze.

EXEMPLU:

COS 25+10 ≡ COS (25+10)
COS 25+10 ≡ (COS 25+10)

11.4.1. Operatorii aritmetici şi logici

În acest paragraf se descriu operatorii aritmetici şi logici cu care se pot defini expresii. Deoarece simbolurile operatorilor constituie separatori de cuvinte, spaţiul între termeni şi operatori este opţional.

Excepţie face operatorul de împărţire (/) care trebuie precedat şi urmat de spaţiu.

Operatorul +

nr1 + nr2

Operatorul „+", calculează suma celor doi termeni nr1, nr2 specificaţi. Termenii pot fi numere întregi sau reale.

EXEMPLU:

```
PRINT 7 +9.652
16.652
```

Operatorul —

nr1 — nr2

Operatorul „—", calculează diferenţa între cei doi termeni nr1 şi nr2. Dacă termenul nr1 este omis şi nu se lasă spaţiu între „—" şi nr2, se generează numărul nr2 cu semn schimbat.

Termenii pot fi numere întregi sau reale.

EXEMPLU:

```
PRINT 8 — 2
6
PRINT 8—2
6
PRINT —3 — —2
—1
```

Trebuie avut grijă în Interpretarea simbolului „—", ca operator sau ca semn minus.

EXEMPLU:

```
7— 1  — generează valoarea 6
7 — 1 — generează valoarea 6
7—  1 — generează valoarea 6
7 —1  — reprezintă o pereche de numere formată din 7 şi —1 (şi nu operaţia 7—1)
```

Operatorul *

nr1 * nr2

Operatorul „*" calculează produsul între cei doi factori.
Factorii pot fi numere întregi sau reale.

EXEMPLU:

```
PRINT 1 * 2 * 3
```

```
      6
PRINT 2 * —1.46
—2.92
```

Operatorul /

nr1 / nr2

Operatorul „/", calculează cîtul între cei doi factori. Factorii pot fi numere întregi sau reale, rezultatul fiind întotdeauna reprezentat în real. Împărțitorul nr2 trebuie să fie diferit de zero. Între operator și factori, trebuie lăsat cîte un spațiu.

EXEMPLU:

```
PRINT  6 / 2
3.0
PRINT 5 / 7.6
0.657895
PRINT 0 / 3
0
PRINT 7 / 0
, , , can't divide by zero
(emite mesajul că nu poate împărţi la zero)
```

Operatorul <

nr1 < nr2

Operatorul „<" (mai mic), generează valoarea logică TRUE, dacă numărul nr1 este strict mai mic decît nr2. În caz contrar generează valoarea logică FALSE.

EXEMPLU:

```
PRINT 2<3                 — generează valoarea logică TRUE
TRUE
PRINT —1.25<—4.65         — generează valoarea logică FALSE
FALSE
```

Operatorul =

obiect 1 = obiect 2

Operatorul „=" (egal), generează valoarea logică TRUE, dacă obiectele obiect 1 și obiect 2 sînt două numere egale, două cuvinte identice sau două liste identice. În caz contrar generează valoarea logică FALSE.

EXEMPLU:

```
PRINT     50   = 2 * 25         — generează valoarea logică TRUE
TRUE
PRINT     7.0  = 7              — generează valoarea logică TRUE
TRUE                            (numărul real este echivalent cu corespondentul întreg)
PRINT     60   =70—10
TRUE
"PIONIER       = WORD "PION "IER — generează valoarea logică TRUE
TRUE
PRINT     60   =70 —10
FALSE
you don 't say what to do with —10
```

În expresia de mai sus 70 —10 nu reprezintă diferenţa între 70 şi 10, ci reprezintă o pereche de numere 70 şi —10. Se evaluează, dacă 60 = 70 şi se generează valoarea logică FALSE, după care se pune întrebarea ce trebuie făcut cu —10.

Operatorul >

nr1 > nr2

Operatorul „>" (mai mare), generează valoarea logică TRUE, dacă numărul nr1 este strict mai mare decît nr2.
În caz contrar, generează valoarea logică FALSE.

EXEMPLU:

```
PRINT 4>3
TRUE
PRINT —15>—8
FALSE
```

11.4.2.. Primitive pentru operaţii matematice

În acest paragraf se vor descrie primitivele cu care utilizatorul poate specifica operaţii matematice.

Primitivele care vor fi tratate în acest paragraf sînt:

ARCCOS	PRODUCT
ARCCOT	RANDOM
ARCSIN	REMAINDER
ARCTAN	ROUND
COSINE	SINE
COTANGENT	SQRT
DIV	SUM
INT	TANGENT

Operaţia ARCCOS

ARCCOS nr

Operaţia ARCCOS calculează valoarea, în grade, a funcţiei arccosinus de argument „nr".

EXEMPLU:

```
PRINT ARCCOS 0.45
63.256316
PRINT ARCCOS 1
90
```

Operaţia ARCCOT

ARCCOT nr

Operaţia ARCCOT calculează valoarea, în grade, a funcţiei arccotangent de argument „nr".

EXEMPLU:

```
PRINT ARCCOT 1
45
```

Operaţia ARCSIN

ARCSIN nr

Operaţia ARCSIN calculează valoarea, în grade, a funcţiei arcsinus de argument „nr".

EXEMPLU:

```
PRINT ARCSIN 0.45
26.743684
```

Operaţia ARCTAN

ARCTAN nr

Operaţia ARCTAN calculează valoarea, în grade, a funcţiei arctangent de argument „nr".

EXEMPLU:

```
PRINT ARCTAN 1
45
```

Operaţia COSINE

COSINE gr
COS gr

Operaţia COSINE calculează valoarea funcţiei cosinus de argument „gr". Argumentul „gr" se specifică în grade.

EXEMPLU:

```
PRINT COS 60
0.5
```

Operaţia COTANGENT

COTANGENT gr
COT gr

Operaţia COTANGENT calculează valoarea, funcţiei cotangent de argument „gr". Argumentul „gr" se specifică în grade.

EXEMPLU:

```
PRINT COT 45
1
```

Operaţia DIV

DIV nr1 nr2

Operaţia DIV calculează cîtul împărţirii, în care nr1 este deîmpărţitul, iar nr2 este împărţitorul. Împărţitorul nr2 trebuie să fie diferit de zero (nr2$\neq$0).

Operaţia este asemănătoare cu operatorul „/".

EXEMPLU:

```
PRINT  DIV 78 2
39
PRINT  DIV —18.64 2
—9.32
PRINT  DIV 4 0
,,,can't divide by zero (nu se poate împărţi la zero)
```

Operaţia INT

INT nr

Operaţia INT generează partea întreagă, prin trunchiere, a argumentului „nr". Dacă argumentul este număr întreg, operaţia INT nu are nici un efect.

EXEMPLU:

```
PRINT INT 6.1279
6
PRINT INT 6.9321
6
PRINT INT 6
6
PRINT INT  —12.3
—12
```

Operaţia PRODUCT

PRODUCT nr1 nr2
(PRODUCT nr1 nr2 nr3 ... nrm)

Operaţia PRODUCT calculează produsul între factorii nr1, nr2 ... nrm. Cînd sînt mai mult de doi factori, este necesară utilizarea parantezelor ( ).

EXEMPLU:

```
PRINT PRODUCT 5 8
40
(PRINT PRODUCT 1 2 3 4 5)
120
```

Operaţia RANDOM

RANDOM nri

Operaţia RANDOM generează un număr întreg pozitiv, aleator, cuprins între 0 şi nri —1. Argumentul „nri" trebuie să fie un număr întreg pozitiv (sau real pozitiv)

EXEMPLU:

```
RANDOM 5
— generează oricare din valorile 0, 1, 2, 3, 4.
```

Trebuie avut în vedere că evaluarea expresiilor:

1+RANDOM 7
RANDOM 7 + 1
— se face în mod diferit şi anume:

a) — în primul caz la numărul aleator generat între 0 şi 6, se adună valoarea 1;

b) — în al doilea caz se adună 7 cu 1 şi apoi se generează un număr aleator între 0 şi 7.

1 + RANDOM 7 ≡ (RANDOM 7) + 1

Operaţia REMAINDER

REMAINDER nr1 nr2

Operaţia REMAINDER calculează restul împărţirii argumentelor nr1 la nr2. Dacă argumentele nu sînt întregi atunci ele se trunchiază şi apoi se calculează restul împărţirii. Argumentul nr2 trebuie să fie diferit de zero.

EXEMPLU:

```
PRINT REMAINDER  17 4
1
PRINT REMAINDER  17 8
1
PRINT REMAINDER —12 5
—2
```

Operaţia ROUND

Round nr

Operaţia ROUND generează cel mai apropiat întreg faţă de argumentul „nr“. Dacă argumentul este egal depărtat de valoarea întreagă superioară lui şi valoarea întreagă inferioară lui, atunci se generează întregul a cărui valoare absolută este mai mare.

EXEMPLU:

```
PRINT ROUND   7.4896
7
PRINT ROUND   7.5087
8
PRINT ROUND   7.5
8
PRINT ROUND  —7.5
—8
```

Operaţia SINE

SINE gr
SIN gr

Operaţia SINE calculează valoarea funcţiei sinus de argument „gr“. Argumentul „gr“ se specifică în grade.

EXEMPLU:

```
PRINT SINE 30
0.5
```

Operaţia SQRT

SQRT nr

Operaţia SQRT (rădăcina pătrată), calculează valoarea funcţiei radical de ordinul 2 din argumentul „nr“. Argumentul „nr“ trebuie să fie pozitiv.

EXEMPLU:

```
PRINT SQRT 36
6
PRINT SQRT 259
16.0935
PRINT SQRT 3 * 3 +4 * 4
5
```

Operaţia SUM

SUM nr1 nr2
(SUM nr1 nr2 ... nrm)

Operaţia SUM calculează suma termenilor nr1 nr2 ... nrm. Cînd sînt mai mult de doi termeni este necesară utilizarea parantezelor ( ).

EXEMPLU:

```
PRINT SUM 2 7
9
PRINT (SUM 4 7 —1 9 —2)
17
```

Operaţia TANGENT

TANGENT gr
TAN gr

Operaţia TANGENT, calculează valoarea funcţiei tangent de argument „gr". Argumentul „gr" se exprimă în grade.

EXEMPLU:

```
PRINT TANGENT 50
1.1917536
```

11.4.3. Primitive pentru operaţii logice

În acest paragraf se descriu primitivele care permit descrierea funcţiilor gice.

Există trei primitive, care formează sistem complet de funcţii (cu ajutorul lor se poate descrie orice funcţie logică) şi anume: AND, NOT, OR.

Tabelele de adevăr a acestor operatori logici sînt prezentate în Tabelul 11.1.

Tabelul 11.1 — Tabelele de adevăr a funcţiilor AND, NOT, OR

arg1	arg2	AND	arg	NOT	arg1	arg2	DR
FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE
FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	TRUE
ΓRUE	FALSE	FALSE			TRUE	FALSE	TRUE
TRUE	TRUE	TRUE			TRUE	TRUE	TRUE

Se defineşte noţiunea de predicat ca fiind o funcţie care generează ca rezultat una din valorile logice TRUE sau FALSE.

Putem considera că operaţiile AND, OR, NOT necesită ca intrări predicate, rezultatele produse de ele fiind tot predicate.

Operaţia AND

AND pred1 pred2
(AND pred1 pred2... predn)

Operaţia AND generează valoarea logică TRUE numai dacă toate argumentele (predicatele) de intrare au valoarea logică TRUE. În caz contrar generează valoarea logică FALSE. (Calculează funcţia SI între argumentele de intrare) (Tabelul 11.1). Dacă sînt mai mult de două argumente, atunci este necesară utilizarea parantezelor ( ).

EXEMPLU:

```
PRINT  AND TRUE TRUE
TRUE
SETPC  4
PRINT  AND PENCOLOUR=2 TRUE
FALSE
```

Operaţia NOT

NOT pred

Operaţia NOT generează valoarea logică TRUE, dacă argumentul (predicatul) de intrare este FALSE, iar dacă argumentul este TRUE generează valoarea logică FALSE.

EXEMPLU:

```
PRINT NOT 2>3
TRUE
```

Operaţia OR

OR pred1 pred2
(OR pred1 pred2 ... predn)

Operaţia OR generează valoarea logică FALSE, dacă toate argumentele (predicatele) de intrare au valoarea logică FALSE. În caz contrar generează valoarea logică TRUE (Tabelul 11.1). Dacă sînt mai mult de două argumente, atunci este necesar să se utilizeze parantezele ( ).

EXEMPLU:

```
PRINT OR FALSE TRUE
TRUE
```

11.5. Primitive LOGO pentru lucrul cu cuvinte şi liste

În acest capitol se descriu primitive care permit lucrul, cu două tipuri de obiecte şi anume:

— lucrul cu cuvinte;
— lucrul cu liste.

Cuvîntul este un obiect LOGO, constituit dintr-un şir de elemente de tip caracter. Un cuvînt LOGO se specifică printr-un şir de caractere prefixat de ghilimele (").

EXEMPLU:

```
PRINT "CUVÎNT
CUVÎNT
```

Delimitarea cuvintelor se face cu ajutorul caracterelor speciale: (spaţiu), (,), [,], <, > +, −, *, /.

Caracterul special ghilimelele (")

— este utilizat pentru a specifica faptul că şirul de caractere ce urmează este un cuvînt.

Caracterul special două puncte (:)

— este utilizat înaintea unui şir de caractere pentru a preciza că se lucrează cu conţinutul variabilei specificată prin cuvîntul respectiv.

Există un cuvînt vid, specificat de ghilimele (") urmate de spaţiu, care este un cuvînt fără nici un caracter (element).

Dacă dorim ca în cadrul unui cuvînt să apară unul din semnele speciale precizate anterior şi acesta să nu fie interpretat ca delimitator trebuie ca să fie precedat de semnul (\) (slash invers SS D).

Lista este un obiect LOGO, constituit dintr-un şir de cuvinte sau de subliste delimitat de paranteze drepte [ ].

EXEMPLU:

```
PRINT [ACEASTA ESTE O LISTĂ]
ACEASTA ESTE O LISTĂ
```

Lista vidă se specifică cu [ ], fără nici un caracter sau cuvînt în interior.

Cuvintele sau sublistele constituie elemente ale listei.

Astfel, lista: [ACEASTA ESTE O LISTĂ], are patru elemente:

ACEASTA
ESTE
O
LISTĂ

Lista [1 [2 3] [4 [5 6]]] — conţine numai trei elemente, nu şase cum pare la prima vedere, şi anume:

1 — element de tip cuvînt;
[2 3] — element de tip listă;
[4 [5 6]] — element de tip listă compusă.

Lista [[ACEASTA ESTE O LISTĂ] [CARE ARE PROPRIETĂŢILE]], conţine două elemente:

[ACEASTA ESTE O LISTĂ] — element de tip listă;
[CARE ARE PROPRIETĂŢILE] — element de tip listă.

Caracterele cu care lucrează limbajul LOGO se împart în:

— caractere alfanumerice: — litere: (A, B, C, ..., Z);
— cifre: (0, 1, 2, ..., 9);
— caractere speciale: (spaţiu) ! " # $ % & ' ( ) * + − . „ / : ; < = > ? [¦] — { } ~

Pentru a reprezenta un caracter în memoria calculatorului acesta se pune în corespondenţă cu un cod binar pe 7 biţi, conform unui standard internaţional numit ASCII (American Standard Code, for Information Interchange).

Fiecare caracter din memoria calculatorului este reprezentat prin codul ASCII, corespunzător.

11.5.1. Primitive care specifică preluarea unor elemente din cuvinte sau liste

În acest paragraf se descriu operaţiile care permit descompunerea şi specificarea unor elemente din cadrul unor cuvinte sau liste.

Se vor descrie primitivele:

BUTFIRST LAST
BUTLAST ITEM
FIRST

Operaţia BUTFIRST

BUTFIRST obiect
BF obiect

Operaţia BUTFIRST generează obiectul specificat, fără primul element al său. Dacă operaţia se aplică unui obiect vid, se generează un mesaj de eroare.

EXEMPLU:

PRINT BUTFIRST "MARE
ARE
PRINT BUTFIRST [ACEASTA ESTE O LISTĂ]
ESTE O LISTĂ
PRINT BUTFIRST [[ACEASTA ESTE O LISTĂ] [CARE ARE PROPRIETĂŢILE]]
CARE ARE PROPRIETĂŢILE
PRINT BUTFIRST [POPESCU CRISTIAN]
CRISTIAN

Operaţia BUTLAST

BUTLAST obiect
BL obiect

Operaţia BUTLAST generează obiectul specificat, fără ultimul element al său. Dacă operaţia se aplică unui obiect vid, se generează un mesaj de eroare.

EXEMPLU:

PRINT BUTLAST "PIONIERI
PIONIER
PRINT BUTLAST [ACEASTA ESTE O LISTĂ]
ACEASTA ESTE O
PRINT BUTLAST [[ACEASTA ESTE O LISTĂ] [CARE ARE PROPRIETĂŢILE]]
ACEASTA ESTE O LISTĂ
PRINT BUTLAST [POPESCU CRISTIAN]
POPESCU

Operaţia FIRST

FIRST obiect

Operaţia FIRST generează primul element al obiectului specificat. Dacă operaţia se aplică unui obiect vid, se generează un mesaj de eroare. Operaţia FIRST aplicată unui cuvînt generează un singur caracter, iar aplicată unei liste generează un cuvînt sau o listă în funcţie de structura listei specificate ca argument.

EXEMPLU:

```
PRINT "PIONIER
P
PRINT FIRST [PIONIER]
PIONIER
PRINT FIRST [ACEASTA ESTE O LISTĂ]
ACEASTA
PRINT FIRST [[ACEASTA ESTE O LISTĂ] [CARE ARE
            PRIORITĂŢILE]]
ACEASTA ESTE O LISTĂ
PRINT FIRST [POPESCU CRISTIAN]
POPESCU
```

Operaţia LAST

LAST obiect

Operaţia LAST generează ultimul element al obiectului specificat. Dacă operaţia se aplică unui obiect vid, se generează un mesaj de eroare. Operaţia LAST aplicată unui cuvînt generează un singur caracter, iar aplicată unei liste generează un cuvînt sau o listă în funcţie de structura listei specificate ca argument.

EXEMPLU:

```
PRINT LAST "PIONIER
R
PRINT LAST [PIONIER]
PIONIER
PRINT LAST [ACEASTA ESTE O LISTĂ]
LISTA
PRINT LAST [[ACEASTA ESTE O LISTĂ] [CARE ARE
           PROPRIETĂŢILE]]
CARE ARE PROPRIETĂŢILE
```

Operaţia ITEM

ITEM nr obiect

Operaţia ITEM generează elementul a cărui poziţie în cadrul obiectului specificat corespunde numărului „nr".

De exemplu: — dacă nr = 4, operaţia ITEM extrage al patrulea element din cadrul obiectului specificat.

Operaţia ITEM se poate aplica unor obiecte de tip cuvînt sau listă.

Dacă operaţia se aplică unui obiect vid, sau numărul „nr" este mai mare decît numărul de elemente ale obiectului se generează un mesaj de eroare.

EXEMPLU:

PRINT ITEM 4 "PIONIER
N
PRINT ITEM 2 [PIONIER]
not enough items in [PIONIER] **— mesaj de eroare deoarece: lista [PIONIER] are un singur element**
PRINT ITEM 4 [IONESCU POPESCU GEORGESCU PETRESCU MATEESCU]
PETRESCU
PRINT ITEM 2 [[ACEASTA ESTE O LISTĂ] [CARE ARE PROPRIETĂŢILE]]
CARE ARE PROPRIETĂŢILE

11.5.2. Primitive care concatenează cuvinte şi liste

În acest paragraf se descriu operaţiile care concatenează (pune împreună), cuvinte sau liste.

Se vor descrie primitivele:

FPUT SENTENCE
LIST WORD
LPUT

Operaţia FPUT

FPUT obiect listă

Operaţia FPUT generează o nouă listă care se formează prin introducerea obiectului specificat în cadrul operaţiei, ca prim element în cadrul listei specificate ca argument.

EXEMPLU:

FPUT "IONESCU [POPESCU GEORGESCU]
— generează lista [IONESCU POPESCU GEORGESCU]
FPUT [ACEASTA ESTE O LISTĂ] [CARE ARE ELEMENTELE]
— generează lista [[ACEASTA ESTE O LISTĂ], CARE ARE ELEMENTELE]
FPUT "PIONIER [ ]
— generează lista [PIONIER]
FPUT [ACEASTA ESTE O LISTĂ] [ ]
— generează lista: [[ACEASTA ESTE O LISTĂ]]

Operaţia LIST

LIST obiect1 obiect2
(LIST obiect1 obiect2 ... obiectn)

Operaţia LIST, generează o listă care are ca elemente obiectele specificate ca argumente. Trebuie notat faptul că dacă sînt mai mult de două argunente este necesară utilizarea parantezelor.

EXEMPLU:

```
LIST "IONESCU [POPESCU GEORGESCU]
— generează lista: [IONESCU [POPESCU GEORGESCU] ]
(LIST "IONESCU "POPESCU "GEORGESCU)
— generează lista: [IONESCU POPESCU GEORGESCU]
LIST "PIONIER [ ]
— generează lista: [PIONIER [ ]]
LIST [ACEASTA ESTE O LISTĂ] [CARE ARE ELEMENTELE]
— generează lista: [[ACEASTA ESTE O LISTĂ] [CARE ARE
                    ELEMENTELE]]
```

Operaţia LPUT

LPUT obiect lista

Operaţia LPUT generează o nouă listă, care se formează prin introducerea obiectului specificat în cadrul operaţiei, ca ultim element în cadrul listei specificate ca argument.

EXEMPLU:

```
LPUT "IONESCU [POPESCU GEORGESCU]
— generează lista: [POPESCU GEORGESCU IONESCU]
LPUT [URMĂTOARE:] [CARE ARE ELEMENTELE]
— generează lista: [CARE ARE ELEMENTELE [URMĂTOARE:]]
LPUT "PIONIER [ ]
— generează lista: [PIONIER]
LPUT [ACEASTA ESTE O LISTĂ] [ ]
— generează lista: [[ACEASTA ESTE O LISTĂ]]
```

Operaţia SENTENCE

SENTENCE obiect1 obiect2
SE obiect1 obiect2
(SENTENCE obiect1 obiect2 ... obiectn)
(SE obiect1 obiect2 ... obiectn)

Operaţia SENTENCE generează o listă care are ca elemente conţinutul elementelor specificate ca argumente.

Dacă sînt mai mult de două obiecte specificate este necesară utilizarea parantezelor.

EXEMPLU:

```
SENTENCE "POPESCU "CRISTIAN
— generează lista: [POPESCU CRISTIAN]
SENTENCE "IONESCU [POPESCU GEORGESCU]
— generează lista: [IONESCU POPESCU GEORGESCU]
SENTENCE [ACEASTA ESTE O LISTĂ] [CARE ARE ELEMEN-
TELE:]
— generează lista: [ACEASTA ESTE O LISTĂ CARE ARE ELEMEN-
TELE]
SENTENCE "PIONIER [ ]
— generează lista [PIONIER]
(SENTENCE "IONESCU "POPESCU "GEORGESCU)
— generează lista [IONESCU POPESCU GEORGESCU]
```

Reamintim că sînt primitive LOGO, cum ar fi SETPOS [x y], DOT [x y], care necesită ca parametru, o listă. Cînd coordonatele x y sînt valorile unor

variabile :X :Y, nu se poate scrie SETPOS [: X :Y], deoarece se generează un mesaj de eroare.

În acest caz, se foloseşte operaţia SENTENCE pentru a crea o listă cu parametrii respectivi.

EXEMPLU:

```
SETPOS SENTENCE :X :Y
```

Deosebirea între SENTENCE şi LIST constă în faptul că:

— operaţia SENTENCE generează o listă alcătuită cu conţinutul obiectelor specificate;

— operaţia LIST generează o listă alcătuită cu obiecte specificate.

EXEMPLU:

```
SENTENCE [ELEVUL POPESCU] [ARE [NOTE MARI]]
— generează lista: [ELEVUL POPESCU ARE [NOTE MARI]]
LIST [ELEVUL POPESCU] [ARE [NOTE MARI]]
— generează lista: [[ELEVUL POPESCU] [ARE [NOTE MARI]]]
```

Operaţia WORD

WORD cuvînt1 cuvînt2

(WORD cuvînt1 cuvînt2 ... cuvîntn)

Operaţia WORD, generează un cuvînt obţinut prin contatenarea (alipirea) cuvintelor specificate ca argument de intrare.

EXEMPLU:

```
PRINT WORD "ION " ESCU
IONESCU
PRINT (WORD "P "O "P "E "S "C "U)
POPESCU
PRINT "WORD "ANA "\ — "MARIA)
ANA-MARIA
PRINT (WORD "ADRI "ANA "\ TAPUS)
ADRIANA TAPUS
```

Observaţie — În versiunea implementată pe HC-85 primitiva WORD interpretează cuvintele formate din cifre ca numere. În acest caz, cuvintele ce au mai mult de 8 cifre sînt transformate în reprezentarea cu mantisă şi exponent.

Pentru a înţelege mai bine cum lucrează primitivele FPUT, LIST, LPUT, SENTENCE, WORD, vom da cîteva exemple care compară efectele acestora:

Operaţie	Arg. 1	Arg. 2	Ieşire
FPUT	"POPESCU	"CRISTIAN	eroare — arg 2 trebuie să fie listă
LIST	"POPESCU	"CRISTIAN	[POPESCU CRISTIAN]
LPUT	"POPESCU	"CRISTIAN	eroare — arg 2 trebuie să fie listă
SENTENCE	"POPESCU	"CRISTIAN	[POPESCU CRISTIAN]
WORD	"POPESCU	"CRISTIAN	POPESCU CRISTIAN

Operaţia	Arg. 1	Arg. 2	Ieşire
FPUT	"CRISTIAN	[ESTE ELEV]	[CRISTIAN ESTE ELEV]
LIST	"CRISTIAN	[ESTE ELEV]	[CRISTIAN [ESTE ELEV]]
LPUT	"CRISTIAN	[ESTE ELEV]	[ESTE ELEV CRISTIAN]
SENTENCE	"CRISTIAN	[ESTE ELEV]	[CRISTIAN ESTE ELEV]
WORD	"CRISTIAN	[ESTE ELEV]	eroare — arg 2 trebuie să fie cuvînt
FPUT	[POPESCU CRISTIAN]	[ESTE ELEV]	[[POPESCU CRISTIAN] ESTE ELEV]
LIST	[POPESCU CRISTIAN]	[ESTE ELEV]	[[POPESCU CRISTIAN] [ESTE ELEV]]
LPUT	[POPESCU CRISTIAN]	[ESTE ELEV]	[ESTE ELEV [POPESCU CRISTIAN]]
SENTENCE	[POPESCU CRISTIAN	[ESTE ELEV]	[POPESCU CRISTIAN ESTE ELEV]
WORD	[POPESCU CRISTIAN]	[ESTE ELEV]	eroare — argumentele trebuie să fie cuvinte
FPUT	"CRISTIAN	[ ]	[CRISTIAN]
LIST	"CRISTIAN	[ ]	[CRISTIAN [ ]]
LPUT	"CRISTIAN	[ ]	[CRISTIAN]
SENTENCE	"CRISTIAN	[ ]	[CRISTIAN]
WORD	"CRISTIAN	[ ]	eroare

11.5.3. Primitive care examinează cuvinte şi liste

În acest paragraf se descriu primitivele care permit utilizatorului să examineze cuvinte şi liste în vederea obţinerii unor informaţii despre acestea.

Se vor descrie primitivele:

ASCII	LISTP
CHAR	MEMBERP
COUNT	NUMBERP
EMPTYP	WORDP
EQUALP	

Operaţia ASCII

ASCII caracter

Operaţia ASCII generează codul ASCII (American Standard Code for Information Interchange), al caracterului specificat.

EXEMPLU

```
PRINT ASCII "A
65
```

(65 = valoarea în zecimal al codului ASCII (01000001_2) pus în corespondenţă cu caracterul A).

```
PRINT ASCII "B
66
```

Operaţia CHAR

CHAR nr

Operaţia CHAR generează caracterul al cărui cod ASCII este numărul „nr". Numărul „nr", trebuie să fie un întreg cuprins între 32 şi 165. Dacă argumentul specificat în cadrul operaţiei CHAR nu este un cod ASCII valid se va genera un mesaj de eroare.

EXEMPLU:

```
PRINT CHAR 65
A
PRINT CHAR 66
B
PRINT WORD CHAR 65 CHAR 66
AB
```

Operaţia COUNT

COUNT obiect

Operaţia COUNT generează un număr care reprezintă cîte elemente are obiectul (cuvînt sau listă) specificat.

EXEMPLU:

```
PRINT COUNT "POPESCU
7
PRINT COUNT [POPESCU CRISTIAN ESTE ELEV]
4
PRINT COUNT [[POPESCU CRISTIAN] [ESTE ELEV]]
2
```

Operaţia EMPTYP

EMPTYP obiect

Operaţia EMPTYP, generează valoarea logică TRUE, dacă obiectul specificat este vid (nu are nici un element) sau valoarea logică FALSE în caz contrar.

EXEMPLU:

```
PRINT EMPTYP "CRISTIAN
FALSE
PRINT EMPTYP BUTFIRST "A
TRUE
PRINT EMPTYP BUTLAST [ELEV]
TRUE
PRINT EMPTYP ITEM 2 [POPESCU [ ] [ESTE ELEV]]
TRUE
```

Operaţia EQUALP

EQUALP obiect1 obiect2

Operaţia EQUALP generează valoarea logică TRUE, dacă obiectele obiect1 şi obiect2, sînt două numere egale; două cuvinte identice sau două

liste identice. În caz contrar generează valoarea logică FALSE. Această operație este echivalentă cu operatorul (=).

EXEMPLU:

```
PRINT EQUALP "CRISTIAN LAST [POPESCU CRISTIAN]
TRUE
PRINT EQUALP 40 2 * 20
TRUE
PRINT EQUALP [EL ESTE [ELEV]] [EL ESTE ELEV]
FALSE
PRINT EQUALP " [ ]
FALSE          (cuvîntul vid nu este identic cu lista vidă)
```

Operația LISTP

LISTP obiect

Operația LISTP generează valoarea logică TRUE dacă obiectul specificat este o listă, sau valoarea logică FALSE în caz contrar.

EXEMPLU:

```
PRINT LISTP 243
FALSE
PRINT LISTP [243]
TRUE
PRINT LISTP [ ]
TRUE               (lista vidă)
PRINT LISTP "
FALSE          (cuvîntul vid nu este identic cu lista vidă)
PRINT LISTP BUTFIRST [ELEV]
TRUE
```

Operația MEMBERP

MEMBERP obiect lista

Operația MEMBERP generează valoarea logică TRUE, dacă obiectul specificat este un element al listei. În caz contrar, generează valoarea logică FALSE.

EXEMPLU:

```
PRINT MEMBERP "ESTE [ACEASTA ESTE O LISTĂ]
TRUE
PRINT MEMBERP "IONESCU [POPESCU IONESCU GEORGESCU]
TRUE
PRINT MEMBERP "IONESCU [POPESCU [IONESCU GEORGESCU]]
FALSE
PRINT MEMBERP "ESCU [POPESCU IONESCU]
FALSE

TO VOCALA :L
PRINT MEMBERP :L [A E I O U]
END

VOCALA A
TRUE
VOCALA B
FALSE
```

Operaţia NUMBERP

NUMBERP obiect

Operaţia NUMBERP generează valoarea logică TRUE dacă obiectul specificat este număr şi valoarea logică FALSE în caz contrar.

EXEMPLU:

```
PRINT NUMBERP 3.4
TRUE
PRINT NUMBERP [3.4]
FALSE
PRINT NUMBERP 3.1E2
TRUE
PRINT NUMBERP BUTFIRST 8.693
TRUE
PRINT NUMBERP "
FALSE
```

Operaţia WORDP

WORDP obiect

Operaţia WORDP generează valoarea logică TRUE, dacă obiectul specificat este cuvînt şi valoarea logică FALSE, în caz contrar. Trebuie avut în vedere, că în LOGO numerele sînt considerate cuvinte.

EXEMPLU:

```
PRINT WORDP "ELEV
TRUE
PRINT WORDP 473
TRUE
PRINT WORDP [473]
FALSE
PRINT WORDP "
TRUE
PRINT BUTFIRST WORDP [ELEV]
FALSE
```

11.6. Primitive LOGO care lucrează asupra variabilelor

În acest capitol se descrie cum se definesc şi cum se utilizează variabilele în limbajul LOGO.

Prin variabilă înţelegem o zonă de memorie unde se păstrează un obiect LOGO (număr, cuvînt sau listă). Zona de memorie are un nume şi o valoare.

Numele variabilei este simbolul cu care ne adresăm variabilei respective, iar valoarea variabilei este conţinutul specificat de acel simbol (conţinutul obiectului respectiv).

O variabilă se creează cu ajutorul comenzii MAKE sau ca argument de intrare într-o procedură.

Referirea numelui variabilei trebuie precedată de ghilimele, iar referirea conţinutului variabilei trebuie precedată de două puncte.

EXEMPLE:

"ALFA — specifică variabila cu numele ALFA;
:ALFA — specifică conţinutul variabilei cu numele ALFA.

În limbajul LOGO sînt două tipuri de variabile:

a) — variabile locale;
b) — variabile globale.

a) Variabilele locale sînt utilizate ca argumente de intrare în procedură. Ele există numai în cadrul procedurii respective, nu sînt accesibile altor proceduri şi dispar din spaţiul de lucru după ce procedura respectivă şi-a terminat execuţia.

b) Variabilele globale sînt create cu comanda MAKE, la nivel de comandă (nu în cadrul procedurilor) şi sînt accesibile tuturor procedurilor din cadrul spaţiului de lucru.

Comanda MAKE

MAKE nume obiect

Comanda MAKE, atribuie conţinutul obiectului specificat variabilei nume.

EXEMPLE:

```
MAKE "DIMENSIUNE 50
PRINT :DIMENSIUNE
50
MAKE "ANIMAL "ELEFANT
PRINT :ANIMAL
ELEFANT
MAKE "ELEV [POPESCU CRISTIAN]
PRINT :ELEV
POPESCU CRISTIAN
```

În exemplele următoare vom scoate în evidenţă relaţia dintre numele variabilei şi valoarea variabilei:

EXEMPLE:

```
MAKE "CULOARE "ALBASTRU
PRINT "CULOARE
CULOARE
PRINT :CULOARE
ALBASTRU
MAKE "FLOARE "CRIN
PRINT :FLOARE
CRIN
MAKE :FLOARE "ALB
PRINT :CRIN
ALB
```

Conţinutul variabilei FLOARE este cuvîntul CRIN. Dacă conţinutului variabilei FLOARE îi atribuim cuvîntul ALB, este echivalent cu atribuirea cuvîntului ALB variabilei CRIN.

EXEMPLU:

```
MAKE :FLOARE obiect
```

— are acelaşi efect cu: MAKE "CRIN obiect

Operaţia NAMEP

NAMEP cuvînt

Operaţia NAMEP generează valoarea logică TRUE, dacă cuvîntul specificat are atribuită o valoare, adică reprezintă numele variabilei. În caz contrar generează valoarea logică FALSE.

EXEMPLU:

```
PRINT NAMEP "PIONIER
FALSE
MAKE "ELEVI [CRISTIAN ŞI ADRIANA]
PRINT :ELEVI
CRISTIAN ŞI ADRIANA
PRINT NAMEP "ELEVI
TRUE
```

Operaţia THING

THING nume

Operaţia THING generează valoarea asociată numelui specificat. Este utilizată pentru a afla ce valoare este asociată valorii variabilei (un fel de adresare indirectă).

THING "CULOARE — este echivalentă cu :CULOARE

Pentru a afla ce valoare este asociată conţinutului

:CULOARE — ar trebui scris ::CULOARE — sintaxă care nu este permisă în LOGO. Pentru a rezolva acest impediment se foloseşte THING :CULOARE

EXEMPLU:

```
MAKE "CULOARE "ALBASTRU
MAKE "ALBASTRU [CULOAREA PREFERATĂ]
PRINT :CULOARE
ALBASTRU
PRINT THING "CULOARE
ALBASTRU
PRINT THING :CULOARE
CULOAREA PREFERATĂ
PRINT THING THING "CULOARE
CULOAREA PREFERATĂ
```

11.7. Primitive LOGO pentru comunicaţia cu mediul extern

În acest capitol se descriu primitivele prin care utilizatorul poate comunica cu mediul extern. Se prezintă modul în care se pot citi caractere de la tastatură şi se poate scrie pe ecranul calculatorului.

11.7.1. Primitive care citesc informaţii din exterior

În acest paragraf se prezintă operaţiile prin care se verifică dacă s-a tastat un caracter la tastatură, se citeşte un caracter sau un şir de caractere de la tastatură.

Se descriu operaţiile:

KEYP
READCHAR
READLIST

Operaţia KEYP

KEYP

Operaţia KEYP generează valoarea logică TRUE, dacă o tastă (sau o combinaţie de taste) este apăsată şi caracterul tastat nu a fost încă preluat printr-o operaţie READCHAR sau READLIST. Este utilizată pentru sesizarea apăsării unei taste.

Operaţia READCHAR

READCHAR
RC

Operaţia READCHAR, aşteaptă tastarea unui caracter, al cărui cod îl furnizează în momentul tastării acestuia. Caracterul tastat nu este tipărit pe ecran.

EXEMPLU — **programul citeşte de la tastatură şi în funcţie de caracterul tastat efectuează una din operaţiile:**
F = deplasează penelul 5 paşi înainte;
B = deplasează penelul 5 paşi înapoi;
R = roteşte penelul cu 30° dreapta;
L = roteşte penelul cu 30° stînga.

```
TO DESEN
MAKE "CAR READCHAR
IF "CAR = "F [FD 5]
IF "CAR = "B [BK 5]
IF "CAR = "R [RT 30]
IF "CAR = "L [LT 30]
DESEN
END

DESEN
```

Operaţia READLIST

READLIST
RL

Operaţia READLIST citeşte o linie de la tastatură (un şir de caractere terminat cu CR) şi furnizează o listă cu conţinutul tastat. Fiecare caracter care se tastează, pe parcursul introducerii liniei, va fi tipărit pe ecran.

EXEMPLU:

```
PRINT COUNT READLIST
ACEASTA ESTE O LISTĂ INTRODUSĂ DE LA TASTATURĂ
8
TO DISCUŢIE
PRINT [CARE ESTE NUMELE TĂU?]
MAKE "NUME READLIST
PRINT [ÎMI PARE BINE CĂ TE CUNOSC]
PRINT [DOMNULE / ] :NUME
END
DISCUŢIE
CARE ESTE NUMELE TĂU?
IONESCU ŞTEFAN
ÎMI PARE BINE CĂ TE CUNOSC
DOMNULE IONESCU ŞTEFAN
```

11.7.2. Primitive care afişează pe ecranul calculatorului

Descrierea comenzilor prin care se pot afişa mesaje pe ecranul calculatorului.

Se descriu operaţiile:

PRINT
SHOW
TYPE

Comanda PRINT

PRINT obiect
PR obiect
(PRINT obiect1 obiect2 ... obiectn)
(PR obiect1 obiect2 ... obiectn)

Comanda PRINT afişează pe ecran obiectele specificate după care se trece implicit la rînd nou.

Obiectul specificat poate fi număr, cuvînt sau listă. În cazul în care este listă se afişează numai conţinutul respectiv, fără parantezele [ ] care o delimitează.

EXEMPLU:

```
PRINT 273
273
PRINT "CUVÎNT
CUVÎNT
PRINT [ACEASTA ESTE O LISTĂ]
ACEASTA ESTE O LISTĂ
```

Comanda SHOW

SHOW obiect

Comanda SHOW afişează pe ecran obiectul specificat după care se trece implicit la rînd nou. Are acelaşi efect cu comanda PRINT numai că în cazul în care obiectul este listă se afişează şi parantezele [ ] care o delimitează.

EXEMPLU:

```
SHOW 273
273
SHOW "CUVÎNT
CUVÎNT
SHOW [ACEASTA ESTE O LISTĂ]
[ACEASTA ESTE O LISTĂ]
SHOW [A SHOW [B C D]
A
[B C D]
```

Comanda TYPE

TYPE obiect

(TYPE obiect1 obiect2 ... obiectn)

Comanda TYPE afişează pe ecran obiectele specificate fără a se trece implicit la rînd nou. Dacă obiectul specificat este lista, nu se afişează şi parantezele [ ] care o delimitează. Are acelaşi efect cu comanda PRINT numai că după afişarea obiectului nu se mai trece implicit la rînd nou.

EXEMPLU

```
TYPE 273
273
TYPE "CUVÎNT
CUVÎNT
TYPE [ACEASTA ESTE O LISTĂ]
ACEASTAESTEOLISTĂ
TYPE "A TYPE [B C D]
AB C D
```

11.7.3. Primitive pentru generare de sunete

În acest paragraf se descrie o primitivă care permite utilizatorului să genereze sunete de durată şi înălţime specificate ca intrări.

Comanda SOUND

SOUND [durată înălţime]

Comanda SOUND generează un sunet de **durată** şi **înălţime** specificate ca intrări sub formă de listă. Durata este specificată în secunde, iar înălţimea în semitonuri faţă de nota C. Durata este un număr natural cuprins între 0 şi 255, iar înălţimea este un număr întreg cuprins între —62 şi 75.

Pentru specificarea duratei şi înălţimii, în funcţie de valorile unor variabile :D şi :I, trebuie să creăm o listă cu aceste variabile utilizînd operaţia SENTENCE.

EXEMPLU

```
SOUND [1 0]
SOUND SENTENCE :X :Y
```

În continuare prezentăm un exemplu care pune în corespondenţă tastele calculatorului cu sunete.

EXEMPLU:

```
TO SUNETAST
SOUND SENTENCE 0.5 (ASCII READCHAR) — 65
SUNETAST
END
SUNETAST
```

11.8. Primitive LOGO care asigură ramificaţia în program şi care controlează fluxul execuţiei comenzilor

Fluxul de control al instrucţiunilor se referă la ordinea în care se execută acestea.

În mod normal instrucţiunile se execută, una după alta, în ordinea în care au fost aşezate în program.

Uneori este necesar să se schimbe această ordine implicită.

Pot apare următoarele cazuri:

— se execută o instrucţiune sau alta în funcţie de îndeplinirea sau nu a unei condiţii;

— repetarea unei liste de instrucţiuni de un număr de ori;

— întreruperea (terminarea) execuţiei unei proceduri înainte de a ajunge la sfîrşitul ei;

— întreruperea temporară (pe o anumită perioadă de timp) a execuţiei unei proceduri.

În acest paragraf se prezintă primitivele prin care utilizatorul poate schimba fluxul secvenţial de execuţie a instrucţiunilor. Aceste primitive se împart în trei categorii în funcţie de efectul pe care îl au în cadrul programelor, şi anume:

— primitive care asigură ramificaţia în cadrul programului în funcţie de anumite condiţii;

— primitive care produc întreruperea execuţiei procedurilor;

— primitive care asigură execuţia şi repetarea unei liste de instrucţiuni.

11.8.1. Primitive pentru ramificarea condiţionată

În acest paragraf se prezintă primitiva care asigură ramificaţia în program în funcţie de o condiţie specificată. Noţiunea de predicat se referă la o operaţie care generează una din valorile logice TRUE sau FALSE. El reprezintă, îndeplinirea sau neîndeplinirea unor condiţii.

Primitiva IF

IF predicat lista de instrucţiuni1

IF predicat lista de instrucţiuni1 lista de instrucţiuni2

Primitiva IF, testează valoarea predicatului şi:

— dacă valoarea logică, generată de predicat este TRUE se execută lista de instrucţiuni1 şi nu se mai execută lista de instrucţiuni2;

— dacă valoarea logică, generată de predicat este FALSE se execută lista de instrucţiuni2 (dacă aceasta există) şi nu se mai execută lista de instrucţiuni1.

Dacă în urma execuţiei uneia din listele de instrucţiuni, se generează un obiect atunci primitiva IF are rol de operaţie, iar în caz contrar are rol de comandă.

Ex:

```
IF "A=READCHAR [PRINT [ATI TASTAT A]]
               [PRINT [NU ATI TASTAT A]]
```

În acest exemplu primitiva IF este interpretată ca fiind comandă. În funcţie de ceea ce se tastează, se recunoaşte dacă s-a apăsat pe A sau altă literă.

Ex:

```
PRINT IF "A=READCHAR [ATI TASTAT A] [NU ATI TASTAT
                                                A]
```

În acest exemplu primitiva IF este interpretată ca fiind operaţie. În urma execuţiei, se generează lista: [ATI TASTAT A] sau lista [NU ATI TASTAT A].

Comanda PRINT afişează rezultatul operaţiei IF.

11.8.2. Primitive pentru întreruperea procedurilor

În acest paragraf se descriu primitivele care permit terminarea execuţiei unei proceduri fără ca aceasta să fi ajuns la sfirşit. De asemenea se prezintă primitivele prin care se realizează întreruperea temporară a execuţiei unei proceduri.

Se vor descrie următoarele comenzi:

BYE WAIT
OUTPUT TOPLEVEL
STOP

Comanda BYE

BYE

Comanda BYE închide o sesiune de lucru LOGO şi transferă controlul în limbajul BASIC.

Dacă se doreşte reintrarea imediată în limbajul LOGO (înainte de a se fi executat ceva în BASIC), pentru a salva diverse proceduri sau programe LOGO, care sînt utile şi nu au fost în prealabil salvate, se poate realiza acest lucru prin introducerea comenzii RUN în BASIC.

Reîntoarcerea imediată din BASIC în LOGO, se poate realiza şi prin comenzile BASIC:

RANDOMIZE USR 24835 — reîntoarcere care nu modifică spaţiul de lucru LOGO.

sau

RANDOMIZE USR 24832 — reîntoarcere care iniţializează spaţiul de lucru LOGO.

Comanda OUTPUT

OUTPUT obiect
OP obiect

Comanda OUTPUT, generează ca ieşire din procedură, obiectul specificat. Totodată întrerupe execuţia procedurii respective (fără a mai continua pînă la sfîrşitul ei, indicat de END) şi reîntoarce controlul în procedura sau programul care a apelat-o.

Trebuie subliniat faptul că primitiva OUTPUT este o comandă, iar procedura care o conţine devine o operaţie, în sensul că generează un obiect de ieşire care este utilizat în procedura sau programul care a apelat-o.

O procedură care se termină cu END este interpretată ca fiind comandă, iar cea care se termină prin OUTPUT, este interpretată ca operaţie.

Comanda OUTPUT are sens numai în cadrul procedurilor. Nu poate fi specificată în cadrul programelor principale.

EXEMPLU:

```
TO POZIŢIE :CAR :LISTA
IF NOT MEMBERP :CAR :LISTA [OUTPUT 0]
IF :CAR = FIRST :LISTA [OUTPUT 1]
OUTPUT 1 +POZIŢIE :CAR BUTFIRST :LISTA
END
MAKE "VOCALE [A E O U]
PRINT POZIŢIE "E :VOCALE
2
PRINT POZIŢIE "B :VOCALE
0
```

Acest exemplu afişează poziţia caracterului specificat în lista de intrare, precizată. Dacă caracterul nu exista în listă, se afişează valoarea 0.

EXEMPLU:

```
TO MEDIE :X :Y
(:X + :Y)/2
END
PRINT MEDIE 4 6
— generează mesaj de eroare:
you don 't say what to do with 5 in MEDIE
(Nu se spune ce se face cu rezultatul 5 în MEDIE),
```

Comanda STOP

STOP

Comanda STOP are ca efect terminarea (întreruperea execuţiei) procedurii curente (în care se află primitiva STOP) şi reîntoarcerea controlului în procedura sau programul care a apelat-o, (Fig. 11.27).

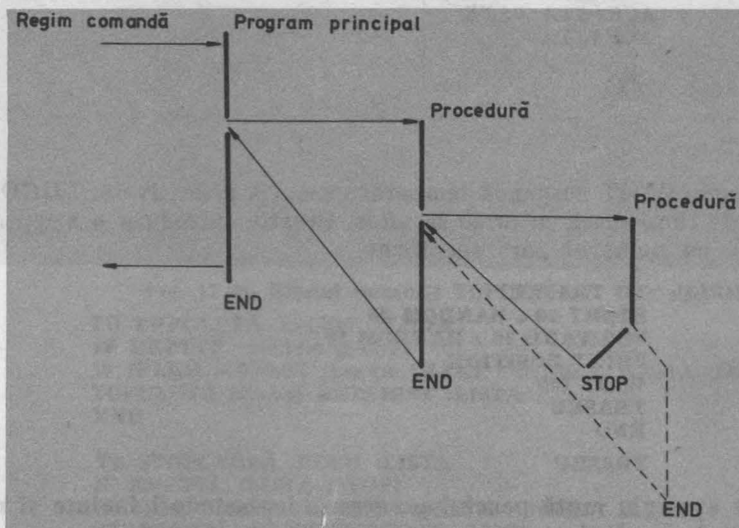


Fig. 11.27. Efectul comenzii STOP

Această comandă are sens numai în cadrul procedurilor.

Trebuie precizat faptul că o procedură care se termină cu STOP este interpretată ca fiind comandă. Ea nu genereazà nici o valoare care ar putea fi utilizată în programul chemător.

EXEMPLU:

```
TO NUMĂRĂ :NR
PRINT :NR
IF :NR=0 [PRINT "GATA STOP]
NUMĂRA :NR —1
END

NUMĂRA 5
5
4
3
2
1
0
GATA

TO EXTRAGE :OBIECT
IF EMPTYP :OBIECT [STOP]
PRINT :OBIECT
EXTRAGE BUTLAST :OBIECT
END
```

```
EXTRAGE "CUVÎNT
CUVÎNT
CUVÎN
CUVÎ
CUV
CU
C
EXTRAGE [ACEASTA ESTE O LISTĂ]
ACEASTA ESTE O LISTĂ
ACEASTA ESTE O
ACEASTA ESTE
ACEASTA
```

Comanda WAIT

WAIT nr

Comanda WAIT suspendă temporar execuţia primitivelor LOGO, pentru o durată de timp egală cu nr/50 secunde. Practic introduce o aşteptare, proporţională cu numărul „nr" specificat.

EXEMPLU:
```
TO TRASEU
RIGHT 10 * RANDOM 30
FORWARD 10 * RANDOM 10
PRINT POSITION
WAIT 100
TRASEU
END
TRASEU
```

Acest exemplu mută penelul pe ecran, deplasîndu-l înainte şi rotindu-l la dreapta în mod aleator.

După fiecare mişcare, se afişează coordonatele poziţiei sale pe o durată de timp de 100/50=2 secunde, după care se reia operaţia de mutare, respectiv afişare.

Comanda TOPLEVEL

TOPLEVEL

Comanda TOPLEVEL are ca efect întreruperea execuţiei procedurii sau programului curent şi reîntoarcerea controlului în regim de comandă, (Fig. 11.28).

Este bine să stabilim care este deosebirea între primitivele TOPLEVEL şi STOP.

Primitiva STOP întrerupe execuţia procedurii curente şi reîntoarce controlul în procedura sau programul care a apelat-o, în timp ce TOPLEVEL întrerupe execuţia întregului program.

În general o comandă TOPLEVEL este utilizată în momentul în care apare o eroare în programul în curs de execuţie (neavînd sens continuarea sa) şi trebuie trecut controlul utilizatorului.

Vom descrie două proceduri care scot în evidenţă deosebirea între TOPLEVEL şi STOP.

EXEMPLU:

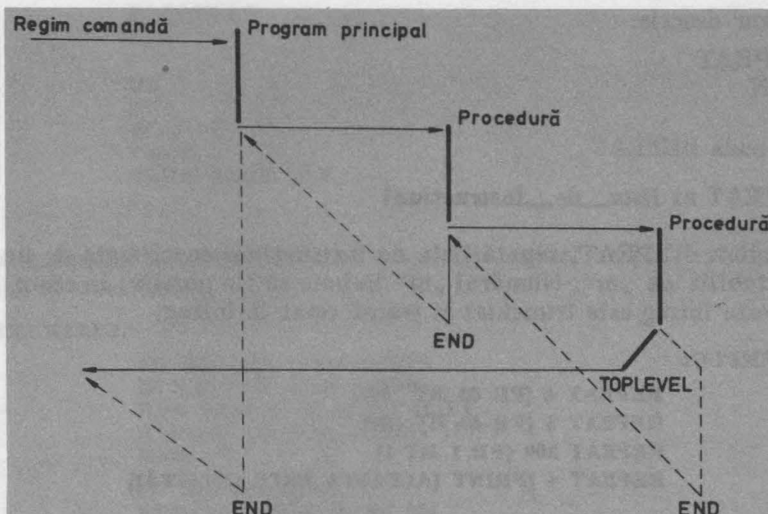


Fig. 11.28. Efectul comenzii TOPLEVEL

```
TO TOPCAUTĂ :ELEM :LISTA
IF EMPTYP :LISTA [STOP]
IF :ELEM = FIRST :LISTA [MAKE "OBIECT :LISTA TOPLEVEL]
TOPCAUTĂ :ELEM BUTFIRST :LISTA
END

TO STOPCAUTĂ :ELEM :LISTA
IF EMPTYP :LISTA [STOP]
IF :ELEM = FIRST :LISTA [MAKE "OBIECT :LISTA]
STOPCAUTĂ :ELEM BUTFIRST :LISTA
END
```

Ambele exemple atribuie variabilei OBIECT, sublista care începe cu elementul specificat de ELEM.

În primul exemplu, procedura atribuie prima sublistă care începe cu elementul specificat, iar în al doilea exemplu, procedura atribuie ultima sublistă.

EXEMPLU:

```
TOPCAUTA "I "CRISTIAN
PRINT :OBIECT
ISTIAN
STOPCAUTA "I "CRISTIAN
PRINT :OBIECT
IAN
TOPCAUTA "ELEVII [ELEVII CLASEI A IV-A SÎNT ELEVII CEI
        MAI BUNI]
PRINT :OBIECT
ELEVII CLASEI A IV-A SÎNT ELEVII CEI MAI BUNI
STOPCAUTA "ELEVII [ELEVII CLASEI A IV-A SÎNT ELEVII CEI
        MAI BUNI]
PRINT :OBIECT
ELEVII CEI MAI BUNI
```

11.8.3. Primitive pentru execuția și repetarea unei liste de instrucțiuni

În acest paragraf se descriu primitivele LOGO, care permit execuția și repetarea unei liste de instrucțiuni specificate.

Se vor descrie:

REPEAT
RUN

Comanda REPEAT

REPEAT nr lista_de_instrucţiuni

Comanda REPEAT, repetă lista de instrucţiuni specificată de un număr de ori stabilit de „nr". Numărul „nr" trebuie să fie pozitiv. În cazul în care „nr" nu este întreg este trunchiat şi transformat în întreg.

EXEMPLU:

```
REPEAT 4 [FD 60 RT  90]
REPEAT 3 [FD 40 RT 120]
REPEAT 360 [FD 1 RT 1]
REPEAT 4 [PRINT [ACEASTA ESTE O LISTĂ]]
```

Comanda RUN

RUN lista_de_instrucţiuni

Comanda RUN execută lista de instrucţiuni specificată ca şi cum ea ar fi fost introdusă în mod direct.

Vom încerca ca în cele două exemple care urmează să evidenţiem cînd o listă este interpretată ca atare şi cînd elementele sale sînt interpretate ca primitive.

EXEMPLU:

```
TO TEXTE
PRINT READLIST
PRINT
TEXTE
END

TO EXECTEXT
PRINT RUN READLIST
PRINT
EXECTEXT
END
```

În primul exemplu lista citită este afişată pe ecran, în timp ce în al doilea exemplu este executată interpretînd elementele ca primitive LOGO şi rezultatul execuţiei este afişat pe ecran.

EXEMPLU:

```
TEXTE
2+3
2+3
6 / 2
6 / 2
42=6 * 7
42=6 * 7
REMAINDER 16 5
REMAINDER 16 5
```

```
EXECTEXT
2 +3
5
6/2
3
42 =6 * 7
TRUE
REMAINDER 16 5
1
```

În exemplul următor, se arată cum o procedură se poate aplica fiecărui element al unei liste.

EXEMPLU:

```
TO REPCOM :COM :LISTA
IF EMPTYP :LIST [STOP]
RUN LIST :COM FIRST :LIST
REPCOM :COM BUTFIRST :LIST
END
TO PATRAT :L
REPEAT 4 [FD :L RT 90]
END
```

REPCOM "PĂTRAT [10 20 30 40 50 60 70], va genera un set de şapte pătrate ale căror laturi sînt specificate în lista specificată, (Fig. 11.29).

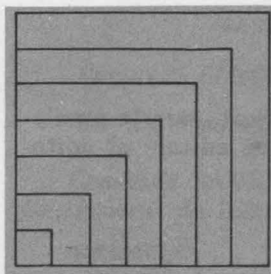


Fig. 11.29. Exemplu de utilizare a comenzii RUN

11.9. Primitive pentru lucrul cu fişiere

În timpul unei sesiuni de lucru LOGO, programul elaborat, procedurile şi variabilele utilizate sînt create într-o zonă de memorie numită spaţiu de lucru. Informaţia în cadrul spaţiului de lucru este organizată în fişiere.

Fişierele pot fi împărţite în următoarele tipuri:

— fişier LOGO, care conţine programe, proceduri şi variabile elaborate cu ajutorul limbajului LOGO;

— fişier de editare LOGO, care conţine fişierul de lucru al editorului LOGO;

— fişier binar, care conţine o zonă de memorie, a cărei informaţie este interpretată ca date binare sau ca program obiect în limbaj de asamblare.

Fişierele de lucru pot fi salvate pe casetă sau încărcate de pe casetă.

Specificarea numelui fişierului se face utilizînd ghilimelele: **"numefişier** — unde numele fişierului poate avea maxim 7 caractere. Pentru a le deosebi este bine ca fiecare nume de fişier să aibă o extensie în funcţie de tip:

.TXT — fişiere text
.SCR — fişiere ecran
.BIN — fişiere binare
.LOG — fişiere LOGO

11.9.1. Primitive pentru încărcare de pe casetă

În acest paragraf se prezintă primitivele LOGO, care permit utilizatorului încărcarea fişierelor rezidente pe caseta magnetică.

Se descriu primitivele:

LOAD
LOADD
LOADSCR

Comanda LOAD

LOAD "numefişier

Comanda LOAD, încarcă de pe casetă fişierul specificat, care în prealabil a fost salvat cu comanda SAVE, în spaţiul de lucru LOGO. În timpul încărcării se afişează ce proceduri sînt definite în cadrul fişierului. La terminarea încărcării, se trece în regim de comandă.

EXEMPLU:

```
LOAD "FIGURI
PATRAT DEFINED
TRIUNGHI DEFINED
```

Comanda LOADD

LOADD "numefişier

Comanda LOADD încarcă de pe casetă fişierul specificat, care în prealabil a fost salvat cu comanda SAVED, în spaţiul fişierului de editare al editorului LOGO.

EXEMPLU:

```
LOAD "FIGURI
```

Comanda LOADSCR

LOADSCR "numefişier

Comanda LOADSCR încarcă de pe casetă fişierul specificat, care în prealabil a fost salvat cu comanda SAVESCR, în zona de memorie asociată ecranului. Informaţia încărcată de pe casetă, se afişează pe ecran.

11.9.2. Primitive pentru salvarea pe casetă

În acest paragraf se prezintă primitivele LOGO, care permit utilizatorului să salveze fişiere pe casetă.

Se descriu primitivele:

SAVE

SAVED
SAVESCR
SAVEALL

Comanda SAVE

SAVE "numefişier [lista de proceduri]

Comanda SAVE salvează pe casetă într-un fişier cu numele numefişier, procedurile specificate în lista de proceduri. Procedurile specificate trebuie să fie definite în cadrul spaţiului de lucru LOGO.

EXEMPLU:

SAVE "FIGURI [PĂTRAT TRIUNGHI]

Comanda SAVEALL

SAVEALL "numefişier

Comanda SAVEALL, salvează pe casetă, într-un fişier cu numele numefişier, tot spaţiul de lucru LOGO. În fişierul respectiv se salvează toate procedurile şi toate variabilele globale.

EXEMPLU:

SAVEALL "FIGURI

Comanda SAVED

SAVED "numefişier

Comanda SAVED, salvează pe casetă, într-un fişier cu numele numefişier, fişierul de lucru al editorului LOGO.

EXEMPLU:

SAVED "EDITSP

Comanda SAVESCR

SAVESCR "numefişier

Comanda SAVESCR, salvează pe casetă, într-un fişier cu numele numefişier, zona de memorie asociată ecranului.

EXEMPLU:

SAVESCR "ECRAN

11.9.3. Primitive pentru controlul imprimantei şi tipărirea informaţiei de pe ecran

În acest paragraf se prezintă primitivele LOGO, care permit utilizatorului să tipărească la imprimantă textul programului elaborat, rezultatele obţinute sau conţinutul ecranului.

Imprimanta trebuie să fie cuplată pe interfaţa paralelă (care deocamdată nu se livrează în configuraţia standard). Imprimanta poate fi conectată pe interfaţa 1 şi în acest caz trebuie instalată înainte de încărcarea şi lansarea în execuţie a limbajului LOGO, cu comenzile din BASIC:

FORMAT "t"; 9600
OPEN #3; "t"

S-a considerat că imprimanta serială funcţionează cu rata de transfer de 9600 bauds.

Se descriu primitivele:

PRINTON
PRINTOFF
COPYSCREEN

Comanda PRINTON

PRINTON

Comanda PRINTON, activează imprimanta. După execuţia acestei comenzi, tot ce se afişează în regim alfanumeric, se tipăreşte la imprimantă (cu excepţia comenzilor în regim comandă).

EXEMPLU:

PRINTON

Comanda PRINTOFF

PRINTOFF

Comanda PRINTOFF, dezactivează imprimanta. După execuţia acestei comenzi, nu se mai tipăreşte la imprimantă textul.

EXEMPLU:

PRINTOFF

Comanda COPYSCREEN

COPYSCREEN

Comanda COPYSCREEN, copiază ecranul la imprimantă. Această primitivă este utilizată în ambele regimuri de lucru ale ecranului, mod alfanumeric sau mod grafic.

Imprimanta conectată la calculator trebuie să fie imprimanta grafică, altfel comanda nu are nici un sens. Această comandă este executată numai cînd imprimanta este conectată prin intermediul interfeţei paralele.

EXEMPLU:

COPYSCREEN

11.9.4. Primitive pentru lucrul cu microdrive-rul

În acest paragraf se prezintă primitivele, care permit utilizatorului să lucreze cu microdriver-ul. Primitivele referitoare la microdrive au aceeaşi formă şi pentru discul flexibil.

Se descriu primitivele:

SETDRIVE
CATALOG
ERASEFILE

Comanda SETDRIVE

SETDRIVE nr

Comanda SETDRIVE, stabileşte ce dispozitiv de memorare extern se utilizează. Dacă „nr" este 0, atunci se consideră caseta (caseta se consideră şi în mod implicit), iar dacă „nr" este un număr între 1 şi 8 stabileşte ca dispozitiv microdriver-ul (unitatea de disc flexibil) cu numărul specificat.

EXEMPLU:

```
SETDRIVE 1
SAVEALL "FIGURI
```

Comanda CATALOG

CATALOG

Comanda CATALOG, afişează numele microdriver-ului (discului) şi numele tuturor fişierelor rezidente pe el. De asemenea, afişează spaţiul liber în kilo-octeţi.

EXEMPLU:

```
SETDRIVE 1
CATALOG
```

Comanda ERASEFILE

ERASEFILE "numefişier.extensie

Comanda ERASEFILE, şterge fişierul, al cărui nume este specificat în cadrul comenzii, de pe microdriver-ul (discul) respectiv. Stabilirea microdriver-ului se face printr-o comandă SETDRIVE. Dacă nu se specifică extensia numelui fişierului, atunci ea se consideră implicit .LOG

Alte extensii care trebuie explicitate sînt:

BIN – specifică fişier obiect (binar)
SCR — specifică fişier asociat zonei ecran
TXT — specifică fişier de lucru al editorului LOGO

EXEMPLU:

```
ERASEFILE "DESEN. SCR
```

11.10. Primitive pentru gestiunea spaţiului de lucru

În acest capitol se prezintă primitivele, care gestionează spaţiul de lucru din memoria calculatorului. Aceste primitive permit afişarea procedurilor din spaţiul de lucru, ştergerea lor precum şi examinarea spaţiului de lucru.

11.10.1. Primitive pentru examinarea spaţiului de lucru

În acest paragraf se prezintă primitivele, care permit utilizatorului să actualizeze spaţiul de lucru, în sensul că poate să obţină informaţii despre spaţiul disponibil şi poate compacta spaţiul utilizat.

Se descriu primitivele:

NODES
RECYCLE

Operaţia NODES

NODES

Operaţia NODES, generează un număr care specifică cîte noduri mai pot fi introduse în spaţiul de lucru continuu. Un nod ocupă 5 octeţi în memorie. Operaţia NODES furnizează utilizatorului cîtă memorie mai este disponibilă spaţiului de lucru pentru proceduri, variabile şi program.

Informaţia furnizată de operaţia NODES este mai exactă, dacă este specificată după o comandă RECYCLE.

Comanda RECYCLE

RECYCLE

Comanda RECYCLE, efectuează o compactare a spaţiului de lucru, prin rearanjarea procedurilor în memoria calculatorului, astfel încît toate spaţiile disponibile, individuale (rămase din ştergerea unor proceduri) să fie adunate într-un spaţiu disponibil continuu şi unic.

Această compactare, se realizează în mod implicit la umplerea spaţiului de lucru. Compactarea durează cca. 1 secundă.

Utilizarea comenzii RECYCLE, previne compactarea implicită care ar putea avea loc într-un moment nedorit din punct de vedere al execuţiei programului.

EXEMPLU:

```
RECYCLE
NODES
```

11.10.2 Primitive pentru afişarea conţinutului spaţiului de lucru

În acest paragraf se prezintă primitivele, care permit utilizatorului, să afişeze informaţii din spaţiul de lucru.

Se descriu primitivele:

PO POPS
POALL POTS
PONS

În timpul afişării, se poate controla defilarea („scroll") informaţiei pe ecran, cu ajutorul funcţiei de control SS S.

Comanda PO

PO lista de nume proceduri

Comanda PO, afişează definiţiile (conţinutul) procedurilor specificate în lista de nume proceduri.

EXEMPLU:

```
PO [PĂTRAT]
TO PĂTRAT :L
REPEAT 4 [FD :L RT 90]
END
```

Comanda POALL

POALL

Comanda POALL, afişează conţinutul tuturor procedurilor şi valorile variabilelor din spaţiul de lucru.

EXEMPLU:

```
POALL
```

Comanda PONS

PONS

Comanda PONS, afişează numele şi valorile tuturor variabilelor din spaţiul de lucru.

EXEMPLU:

```
PONS
MAKE "F 3
MAKE "LIST [A B C]
```

Comanda POPS

POPS

Comanda POPS, afişează conţinutul tuturor procedurilor din spaţiul de lucru.

EXEMPLU:

```
POPS
```

Comanda POTS

POTS

Comanda POTS, afişează numele şi parametrii de intrare ai tuturor procedurilor din spaţiul de lucru.

EXEMPLU:

```
POTS
```

11.10.3. Primitive pentru ştergerea informaţiilor din spaţiul de lucru

În acest paragraf se prezintă primitivele care şterg informaţiile din spaţiul de lucru.

Se descriu primitivele:

ERALL ERNS
ERASE ERPS
ERN

Comanda ERALL

ERALL

Comanda ERALL, şterge toate procedurile şi variabilele din spaţiul de lucru. Are ca efect ştergerea întregului spaţiu de lucru. Este echivalent cu a începe o nouă sesiune de lucru LOGO. Înainte de a specifica o astfel de comandă trebuie avut grijă ca în prealabil să se salveze procedurile utile.

Conţinutul spaţiului de editare nu este afectat de comanda ERALL. Pentru a şterge şi spaţiul de editare se foloseşte combinaţia.

ERALL
EDIT

Comanda ERASE

ERASE lista_de_nume_proceduri

ER lista_de_nume_proceduri

Comanda ERASE, şterge din spaţiul de lucru, procedurile specificate în **lista_de_nume_proceduri**.

Dacă se specifică o singură procedură, atunci ea se poate preciza sub forma de listă sau cuvînt.

EXEMPLU:

```
ERASE [PĂTRAT TRIUNGHI]
ERASE "PĂTRAT
ERASE [PĂTRAT]
```

Comanda ERN

ERN lista_de_nume_variabile

Comanda ERN, şterge din spaţiul de lucru, variabilele specificate în **lista_de_nume_variabile**. Dacă se specifică o singură variabilă, atunci ea se poate preciza ca listă sau cuvînt.

EXEMPLU:

```
ERN "L
ERN [L]
ERN [L A]
```

Comanda ERNS

ERNS

Comanda ERNS, şterge din spaţiul de lucru, toate variabilele care au fost definite pînă în momentul respectiv.

EXEMPLU:

ERNS

Comanda ERPS

ERPS

Comanda ERPS, şterge din spaţiul de lucru, toate procedurile.

EXEMPLU:

ERPS

11.11. Primitive pentru definirea şi editarea procedurilor

În acest capitol se prezintă modul în care utilizatorul îşi pregăteşte programul în limbajul LOGO.

Există două moduri în care se pot defini procedurile:

a) mod editare în care se utilizează editorul LOGO
b) mod interactiv în care procedura se defineşte în regim comandă

În mod editare, orice eroare de introducere poate fi înlăturată imediat cu ajutorul facilităţilor oferite de editorul LOGO, în timp ce în mod interactiv nu se poate reveni asupra textului introdus.

Cel de-al doilea mod este utilizabil numai în cazul procedurilor simple, de complexitate redusă.

11.11.1. Primitive pentru definire şi editare proceduri

În acest paragraf se prezintă primitivele LOGO, care permit utilizatorului să-şi definească şi editeze proceduri.

Se descriu primitivele:

EDIT TO
ENDS END

Comanda EDIT

EDIT "numeprocedura
ED "numeprocedura

Comanda EDIT, lansează în execuţie editorul LOGO în vederea definirii sau editării procedurii cu numele specificat de **numeprocedură**.

Editarea mai multor proceduri, se realizează prin specificarea în loc de nume procedură o listă cu numele procedurilor respective.

Pentru a defini o procedură nouă, se introduce:

EDIT [] sau
EDIT "numeprocedură

În acest caz, se lansează editorul LOGO, care va avea spaţiul său de lucru vid.

Editarea unei proceduri definite anterior, se realizează prin introducerea EDIT "numeprocedură.

Revenirea la ultima procedură cu care a lucrat editorul, care se află deja în spaţiul său de lucru, se face prin introducerea comenzii EDIT

În regim de editare, caracterul de început de rînd (?) din regim comandă nu mai apare.

În cadrul editorului, sînt o serie de caractere de control al editării, care se referă la:

— mutarea cursorului pe ecran;
— ştergerea unei zone de text;
— terminarea unei sesiuni de lucru cu editorul.

Prezentarea caracterelor de control, utilizate în procesul de editare:

EMODE — reprezintă modul extins de utilizare a tastaturii, obţinut prin apăsarea simultană a tastelor:
CS — litere mari (caps shift)
SS — caractere simbolice (symbol shift)

Caractere de control al editării:

— caractere de control pentru mutarea cursorului:

CS 5 — mută cursorul un caracter la stînga (←)
CS 6 — mută cursorul o linie mai jos (↓)
CS 7 — mută cursorul o linie mai sus (↑)
CS 8 — mută cursorul un caracter la dreapta (→)

EMODE CS 5 — mută cursorul la începutul liniei curente
EMODE CS 6 — mută cursorul la sfîrşitul ecranului
EMODE CS 7 — mută cursorul la începutul ecranului
EMODE CS 8 — mută cursorul la sfîrşitul liniei curente

EMODE B — mută cursorul la începutul textului de editare
EMODE E — mută cursorul la sfîrşitul textului de editare
EMODE P — mută cursorul la pagina anterioară
EMODE N — mută cursorul la pagina următoare

Observaţie — **În timpul trecerii la pagina anterioară sau următoare de text procesul de defilare („scroll") poate fi oprit cu SS S pînă la tastarea oricărei taste.**

— caractere de control pentru ştergere text:

CS 0 — şterge un caracter aflat la stînga cursorului;

EMODE Y — şterge linia curentă (cea pe care se găseşte cursorul) începînd cu poziţia în care se află cursorul pînă la sfîrşit şi o memorează într-o zonă de memorie;

EMODE R — readuce linia ştearsă şi salvată cu EMODE Y, începînd cu poziţia curentă a cursorului.

— caractere de control pentru terminarea unei sesiuni de lucru cu editorul.

EMODE C — se trece din regim editare în regim comandă salvînd toate modificările efectuate.

CS BREAK — ignoră sesiunea curentă de editare (nu ia în considerare modificările făcute) şi trece în regim de comandă.

Comanda EDNS

EDNS lista de nume

Comanda EDNS, permite editarea numelor variabilelor şi a valorilor asignate acestora. Sînt listate variabilele specificate în lista de nume şi valorile asociate acestora, permiţînd utilizatorului să editeze numele sau valorile.

Dacă EDNS nu are specificat o listă de nume atunci se listează toate variabilele din spaţiul de lucru cu valorile asignate şi se pot edita în mod corespunzător.

EXEMPLU:

```
EDNS
MAKE "FLOARE "CRIN
MAKE "DIMENSIUNE 50
MAKE "ELEV [POPESCU CRISTIAN]
```

Cu caracterele de control al editării se pot modifica numele sau valorile variabilelor. După modificare se revine în regim comandă cu EMODE C.

Comanda TO

TO nume_procedură intr1 intr2... intrn

Comanda TO, permite definirea unei proceduri în regim comandă. După introducerea comenzii de definire TO urmată de numele procedurii şi de parametrii de intrare în procedură (intr1 intr2 ... intrn), pe ecran va apare un nou caracter de început rînd (>), care ne invită să introducem definirea procedurii.

Comenzile introduse nu sînt executate, ci vor constitui corpul procedurii respective.

Terminarea definirii procedurii, se realizează cu comanda END.

EXEMPLU:

```
? TO PĂTRAT :L
> REPEAT 4 [FORWARD :L RIGHT 90]
> END
PĂTRAT Defined
?
```

Abandonarea definirii unei proceduri, în timpul introducerii acesteia, se realizează prin tastarea CS BREAK simultan. Pentru a schimba conţinutul unei proceduri cu ajutorul comenzii TO trebuie ca aceasta să fie ştearsă în prealabil cu comanda ERASE.

Editarea conţinutului unei proceduri este de preferat să se facă cu EDIT.

Comanda END

END

Comanda END are un rol special şi anume acela de a specifica sfîrşitul unei proceduri. Ea trebuie pusă în ultima linie de definire a procedurii.

11.11.2. Primitive pentru modificarea procedurilor sub controlul programului

În acest paragraf se prezintă primitivele care permit definirea procedurilor sub controlul programului.

Se descriu primitivele:

COPYDEF PRIMITIVEP
DEFINE TEXT
DEFINEP

Comanda COPYDEF

COPYDEF numenou nume

Comanda COPYDEF copiază corpul de definire (conţinutul) procedurii cu numele **nume** (care a fost definită anterior), într-o nouă procedură, cu numele **numenou**. Procedura existentă iniţial nu îşi pierde conţinutul.

Ex:

COPYDEF "PT "PĂTRAT

Copiază definiţia procedurii PĂTRAT, sub numele PT. În urma execuţiei comenzii, vom avea două proceduri, una cu numele PT şi alta cu numele PĂTRAT, care au acelaşi conţinut.

Comanda DEFINE

DEFINE nume lista

Comanda DEFINE, permite generarea prin program a corpului unei proceduri. **Nume** reprezintă numele procedurii astfel create, iar **lista** specifică parametrii de intrare în procedură şi liniile de definire a acesteia. Dacă procedura nu are parametri de intrare, trebuie ca lista să înceapă cu o sublistă vidă.

EXEMPLU:
```
DEFINE "DREPTUNGHI [[ :L :I] [FD :I RT 90] [FD :L RT 90]
                    [FD :I RT 90] [FD :L RT 90]]
PO "DREPTUNGHI
```

```
TO DREPTUNGHI :L :I
FD :I  RT 90
FD :L  RT 90
FD I   RT 90
FD :L  RT 90
END
```

```
DEFINE "SALUT [[] PRINT [BUNĂ ZIUA ]
PO "SALUT
TO SALUT
PRINT [BUNĂ ZIUA]
END
```

Operaţia DEFINEP

DEFINEP cuvînt

Operaţia DEFINEP, generează valoarea logică TRUE, dacă parametrul **cuvînt** este numele unei proceduri definite.
În caz contrar, generează valoarea logică FALSE.

EXEMPLU:

```
PRINT DEFINEP "PĂTRAT
TRUE
```

Operaţia PRIMITIVEP

PRIMITIVEP nume

Operaţia PRIMITIVEP generează valoarea logică TRUE, dacă parametrul **nume** reprezintă o primitivă LOGO.
În caz contrar, generează valoarea logică FALSE.

EXEMPLU:

```
PRINT PRIMITIVEP "FORWARD
TRUE
PRINT PRIMITIVEP "PĂTRAT
FALSE
```

Operaţia TEXT

TEXT nume

Operaţia TEXT furnizează definiţia procedurii, cu numele **nume** ca o listă de subliste, care poate fi utilizată ca intrare într-o comandă DEFINE.

Primul element din listă reprezintă intrările în procedură. Celelalte subliste reprezintă, fiecare, cîte o linie din definiţia procedurii.

EXEMPLU:

```
SHOW TEXT "DREPTUNGHI
[[:L :I] [FD :I RT 90 ] [FD :L RT 90] [FD :I RT 90]
[FD :L RT 90]]
```

11.12. Primitive pentru funcţii diverse

În acest capitol se prezintă primitivele care permit accesul utilizatorului la memoria calculatorului în mod direct. Utilizatorul trebuie să cunoască bine alocarea spaţiului de memorie, deoarece aceste primitive adresează memoria privită ca resursă a calculatorului şi orice modificare a conţinutului său poate distruge conţinutul spaţiului de lucru sau conţinutul variabilelor de sistem.

În general aceste primitive încep cu punct (.).

11.12.1. Primitive pentru accesul la memoria calculatorului

Aceste primitive permit utilizatorului să aibă acces în mod direct la memoria calculatorului.

Se descriu primitivele:

.BLOAD .EXAMINE
.BSAVE .RESERVE
.CALL .RESERVED
.DEPOSIT

Comanda .BLOAD

.BLOAD "numefişier adr

Comanda .BLOAD încarcă de pe casetă sau microdrive (disc) în memoria calculatorului conţinutul fişierului specificat începînd cu adresa „**adr**". Fişierul de pe casetă poate conţine un program obiect direct executabil (de către unitatea de comandă a calculatorului) sau o zonă de date binare.

Trebuie avut grijă ca adresa „adr" să nu se suprapună peste spaţiul de lucru LOGO, sau peste zona variabilelor de sistem.

EXEMPLU:

```
.BLOAD "FOBIECT 64824
```

Conţinutul fişierului obiect FOBIECT se citeşte de pe casetă şi se depune în memorie începînd cu adresa 64824.

Comanda .BSAVE

.BSAVE "numefişier [startadr lng]

Comanda .BSAVE salvează pe casetă sub numele **numefişier**, conţinutul unei zone de memorie delimitată de adresa de început **startadr** şi lungimea **lng**.

EXEMPLU:

```
.BSAVE "FOBIECT [64824 200]
```

În exemplul de mai sus se salvează pe casetă zona de memorie 64824 ... 65024 într-un fişier care are numele FOBIECT.

Comanda .CALL

.CALL adr

Comanda .CALL lansează în execuţie un program obiect, care se află în memoria calculatorului, începînd cu adresa „**adr**". Programul obiect trebuie încărcat în prealabil cu o comandă .BLOAD

EXEMPLU:

```
.CALL 64824
```

Comanda .DEPOSIT

.DEPOSIT adr val

Comanda .DEPOSIT depune în celula de memorie, a cărei adresă este specificată de „**adr**", valoarea „**val**".

EXEMPLU:

```
.DEPOSIT 65010 55
```

Procedura care defineşte un caracter special (1/4), care poate fi ulterior utilizat în cadrul programelor LOGO, se realizează astfel:

EXEMPLU:

```
TO USR.A
.DEPOSIT 65368 64
.DEPOSIT 65369 68
.DEPOSIT 65370 72
.DEPOSIT 65371 80
.DEPOSIT 65372 42
.DEPOSIT 65373 74
.DEPOSIT 65374 15
.DEPOSIT 65375 2
PRINT CHAR 144
END
USR.A
1/4
```

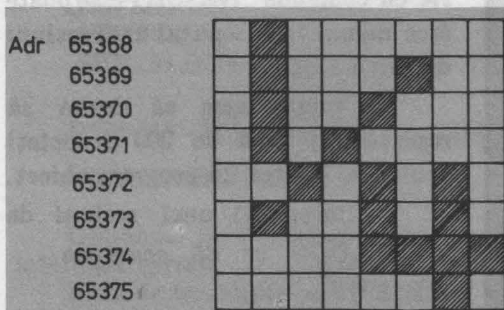


Fig. 11.30. Exemplu de utilizare a comenzii .DEPOSIT

Un caracter este definit printr-o matrice de puncte. Avînd adresa de început a matricei respective, se poate modifica în mod corespunzător forma caracterului în funcţie de ceea ce se depune în acea zonă. Valorile depuse trebuie interpretate ca valori binare. Biţii egali cu 1 se consideră puncte aprinse, iar biţii egali cu 0 se consideră puncte stinse.

În exemplul precedent s-a obţinut desenul din Fig. 11.30.

Adresă	Conţinut				
	Zecim.		Hexazecim.		Binar
65368	64	=	40	=	01000000
65369	68	=	44	=	01000100
65370	72	=	48	=	01001000
65371	80	=	50	=	01010000
65372	42	=	2A	=	00101010
66373	74	=	4A	=	01001010
65374	15	=	0F	=	00001111
65375	2	=	02	=	00000010

Tabelul 11.2 Conţinutul zonei de memorie 65368–65375 ca urmare a comenzii .DEPOSIT

Operaţia .EXAMINE

EXAMINE adr

Operaţia .EXAMINE generează o valoare ce reprezintă conţinutul celulei de memorie specificată de adresa **adr**.

EXEMPLU:

```
PRINT .EXAMINE 65370
72
```

Comanda .RESERVE

.RESERVE nr

Comanda .RESERVE rezervă o zonă de memorie de lungime „nr", la sfîrşitul spaţiului de lucru LOGO. Această zonă devine neutilizabilă sub controlul limbajului LOGO (cu excepţia comenzilor .BLOAD, .DEPOSIT), fiind rezervată pentru o eventuală încărcare a unor programe obiect, sau a unor date binare.

	65535	
VARIABILE SISTEM		VARIABILE SISTEM
	65024	ZONA REZERVATA
Spaţiu de lucru LOGO	65824	Spaţiu de lucru LOGO
	RAM	
LOGO		LOGO
	24832	
VARIABILE SISTEM		VARIABILE SISTEM
	16384	
SISTEM (BASIC)	ROM	SISTEM (BASIC)
	0	

Fig. 11.31. Exemplu de utilizare a comenzii .REVERSE

Rezervarea unei zone de memorie cu comanda .RESERVE se poate face numai la începutul unei sesiuni de lucru LOGO.

Să presupunem că dorim să rezervăm o zonă de 200 de octeţi pentru a încerca un program obiect.

La începutul unei sesiuni de lucru LOGO se dă comanda .RESERVE 200.

Structura memoriei calculatorului este prezentată în Fig. 11.31.

Operaţia .RESERVED

.RESERVED

Operaţia .RESERVED generează adresa de început şi adresa de sfîrşit a zonei de memorie care în prealabil a fost rezervată cu o comandă .RESERVE.

EXEMPLU:

PRINT .RESERVED
64824 65024

11.12.2. Primitive pentru lucrul cu interfaţa serială

În acest paragraf se prezintă primitivele care permit utilizatorului să lucreze cu interfaţa serială RS232.

La interfaţa serială se pot conecta echipamente periferice care lucrează serial asincron conform standardului CCITT V. 24. Conexiunea între calculator şi echipament se face pe linie serială. Informaţia paralelă (8 biţi) se trimite/recepţionează pe cîte o singură linie de date (una pentru recepţie alta pentru transmisie).

Structura logică a informaţiei de date este prezentată în figura 11.32.

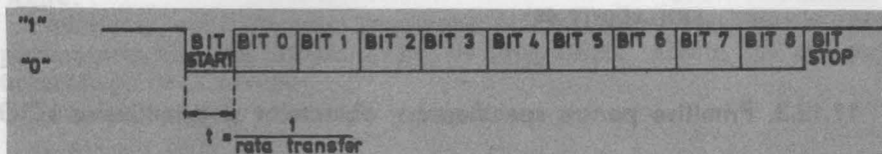


Fig. 11.32. Structura logică a informaţiei seriale

Durata „t", cît un bit este ţinut pe linia de transmisie, este funcţie de rata de transfer a datelor. Rata de transfer este standardizată şi poate fi: 50, 110, 150, 300, 600, 1200, 2400, 4800, 9600, 19200 bauds.

(Baud = unitate de măsură a ratei de transfer, este echivalent într-o oarecare măsură, cu numărul de biţi pe secundă care se transmit).

Se descriu primitivele:

.SETSERIAL
.SERIALIN
.SERIALOUT

Comanda .SETSERIAL

.SETSERIAL ratatransf

Comanda .SETSERIAL stabileşte rata de transfer (viteza de transfer) a interfeţei seriale la valoarea ratatransf.

Rata de transfer poate fi una din valorile: 50, 110, 150, 300, 600, 1200, 2400, 4800, 9600, 19200.

La iniţierea unei sesiuni de lucru LOGO, rata de transfer se stabileşte în mod implicit la valoarea 9600.

Ex:

.SETSERIAL 1200

Stabileşte rata de transfer la 1200 bauds.

Operaţia .SERIALIN

.SERIALIN

Operaţia .SERIALIN furnizează o valoare pe 8 biţi (între 0 şi 255) ce reprezintă informaţia recepţionată de interfaţa serială. Operaţia .SERIALIN aşteaptă pînă cînd echipamentul serial cu care este conectat calculatorul transmite un caracter (un cuvînt pe 8 biţi).

EXEMPLU:

```
PRINT .SERIALIN
65
```

Comanda .SERIALOUT

.SERIALOUT data

Comanda .SERIALOUT transmite, prin intermediul interfeţei seriale, valoarea dată (care trebuie să fie între 0 şi 255) către echipamentul serial care este conectat cu calculatorul.

EXEMPLU:

```
.SERIALOUT 65
```

11.12.3. Primitive pentru specificarea obiectelor şi primitivelor LOGO

În acest paragraf se prezintă comenzile LOGO care permit utilizatorului să afişeze toate primitivele recunoscute de limbajul LOGO sau să afişeze ce primitive şi proceduri sînt implementate la un moment dat în spaţiul de lucru.
Se descriu primitivele:
.CONTENTS
.PRIMITIVES

Operaţia .CONTENTS

.CONTENTS

Operaţia .CONTENTS generează o listă cu tot ce este implementat în spaţiul de lucru LOGO. Lista cuprinde procedurile utilizatorului, variabilele etc.

Trebuie avut în vedere că operaţia .CONTENTS poate utiliza un spaţiu de memorie considerabil de mare.

Comanda .PRIMITIVES

.PRIMITIVES

Comanda .PRIMITIVES afişează toate primitivele implicite, recunoscute de limbajul LOGO.

EXEMPLU:

```
.PRIMITIVES
```

Capitolul 12. **Tehnici de programare în LOGO și aplicații**

12.1. Proiectarea programelor

Pentru scrierea programelor în limbajul LOGO se utilizează primitivele prezentate în capitolul 11. Operaţiile care conduc la rezolvarea unei probleme sînt descrise în general de un algoritm. Algoritmul constă dintr-o mulţime de operaţii executate într-o ordine bine stabilită, asupra unui set de valori denumite date de intrare şi care produc, în timp finit, un set de valori denumite date de ieşire.

Elaborarea unui program presupune atît realizarea algoritmului cît şi implementarea acestuia într-un limbaj de programare. Aceste operaţii pot fi structurate pe două nivele:

— nivelul logic;
— nivelul fizic.

Nivelul logic constă din trei etape:

— etapa de definire;
— etapa de descriere;
— etapa de evaluare.

Nivelul fizic constă din două etape:

— etapa de implementare;
— etapa de testare.

Etapa de definire constă dintr-un set de operaţii prin care se specifică:

— lista variabilelor de intrare şi ieşire ale programului;
— tipul şi organizarea datelor de intrare şi ieşire;
— reprezentarea internă şi externă a datelor.

Etapa de descriere constă în specificarea algoritmului care prelucrează mulţimea valorilor de intrare. Algoritmul poate fi descris în cuvinte din limbajul natural, în pseudocod sau prin organigrame. Se definesc astfel o serie de proceduri de bază, specificate prin:

— setul de valori de intrare;
— setul de valori de ieşire;
— operaţiile asupra datelor.

Etapa de evaluare constă în verificarea faptului că algoritmul elaborat în etapa de descriere realizează toate cerinţele enunţului problemei şi că repre-

zentarea datelor este corect aleasă. Dacă în urma evaluării rezultatele nu sînt satisfăcătoare se reiau etapele de definire şi descriere pînă cînd se realizează toate cerinţele problemei.

Aceste etape spunem că reprezintă nivelul logic deoarece nu depind decît în mică măsură de limbajul concret în care se va face implementarea.

Nivelul fizic constă în implementarea efectivă a algoritmului într-un limbaj de programare adecvat.

Etapa de implementare constă în transpunerea în limbajul de programare ales, în cazul nostru LOGO, a algoritmului prezentat în etapa de descriere.

În *etapa de testare* se validează buna funcţionare a programului. În această etapă se elimină erorile sintactice (nerespectarea regulilor de scriere specifice limbajului ales) şi a celor semantice generate de un algoritm incorect sau o transcriere incorectă a algoritmului în limbajul respectiv.

Se reia faza de implementare pînă se obţin rezultatele dorite.

Respectarea acestor etape creează premisele unei proiectări sistematice a programelor. Limbajul LOGO oferă facilităţi de elaborare sistematică a programelor pornind de la general la particular (top-down) prin rafinări succesive pînă la obţinerea unei versiuni corecte a programului.

EXEMPLU:

În continuare se va exemplifica această tehnică de abordare de la general la particular prin descompunerea unei probleme în subprobleme în cazul unui program de grafică. Se cere să se deseneze un robot cu specificaţiile din fig. 12.1.

În primul rînd descompunem robotul în elementele componente: CAP, GIT, CORP, MIINI, PICIOARE.

La rîndul lor unele din aceste elemente pot fi descompuse în elemente mai simple Astfel CAP-ul poate fi descompus în:

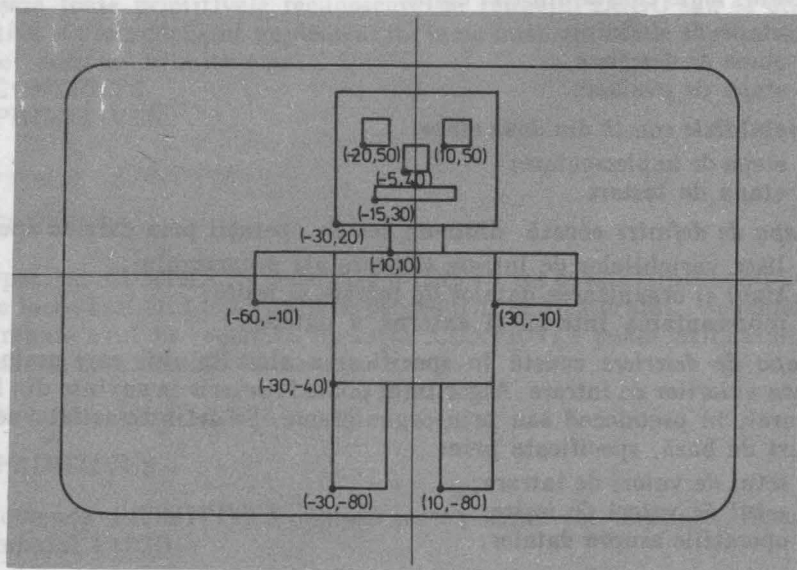


Fig. 12.1. Robot

— formă cap FORMACAP;
— ochi stînga OCHIST;
— ochi dreapta OCHIDR;
— nas NAS;
— gura GURA.

De fapt se observă că robotul are la bază două elemente simple: dreptunghiul şi pătratul.

Astfel pentru a construi robotul este necesar să se definească dreptunghiul şi pătratul cu laturi variabile şi coordonatele unui vîrf. Plecînd de la general la particular, desenarea robotului se va realiza cu următoarea succesiune de proceduri:

```
TO ROBOT
   CAP
   GÎT
   CORP
   MÎINI
   PICIOARE
END

TO CAP
   FORMACAP
   OCHIST
   OCHIDR
   NAS
   GURĂ
END

TO MÎINI
   MINAST
   MINADR
END

TO PICIOARE
   PICIORST
   PICIORDR
END

TO PĂTRAT :X :Y :L
   PU
   SETPOS SENTENCE :X :Y
   PD
   REPEAT 4 [FD :L RT 90]
END

TO DREPTUNGHI :X :Y :I :L
   PU
   SETPOS SENTENCE :X :Y
   PD
   REPEAT 2 [FD :I RT 90 FD :L
      RT 90]
END

TO FORMACAP
   DREPTUNGHI — 30 20 50 60
END

TO OCHIST
   PĂTRAT —20 50 10
END

TO OCHIDR
   PĂTRAT 10 50 10
END

TO NAS
   PĂTRAT —5 40 10
END

TO GURA
   DREPTUNGHI —15 30 5 30
END

TO GÎT
   DREPTUNGHI —10 10 —10 20
END

TO CORP
   DREPTUNGHI —30 —40 50 60
END

TO MINAST
   DREPTUNGHI —60 —10 20 30
END

TO MINADR
   DREPTUNGHI 30 —10 20 30
END

TO PICIORST
   DREPTUNGHI —30 —80 40 20
END

TO PICIORDR
   DREPTUNGHI 10 —80 40 20
END
```

EXEMPLU:

Un alt exemplu care ilustrează această tehnică constă în desenarea unui peisaj similar cu cel din fig. 12.2

```
TO PEISAJ
   CASA
   BRAZI
END

TO CASA
   PEREŢI
   GEAMURI
   UŞA
   ACOPERIŞ
END

TO PEREŢI
   DREPTUNGHI —20 —60 70 120
END
```

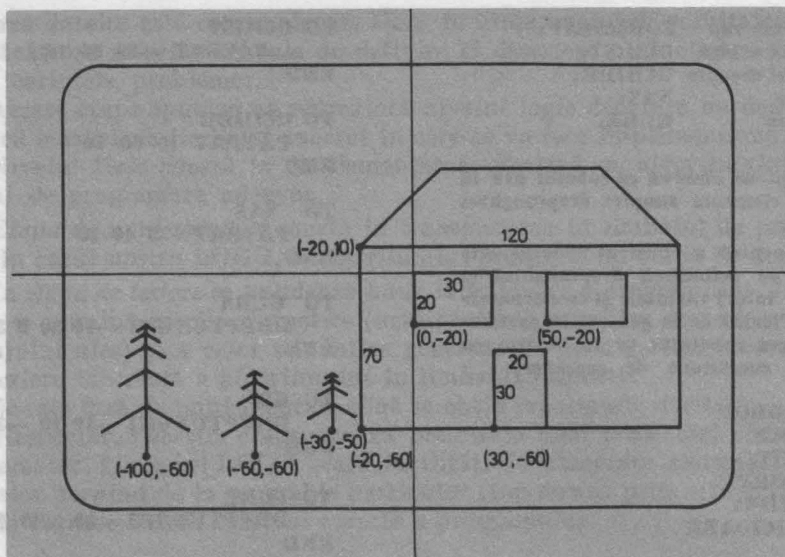


Fig. 12.2. Peisaj

```
TO GEAMURI
   DREPTUNGHI 0 —20 20 30
   DREPTUNGHI 50 —20 20 30
END

TO UŞA
   DREPTUNGHI 30 —60 30 20
END

TO ACOPERIŞ
   PU
   SETPOS [—20 10]
   PD
   RT 45
   FD 30 * 1.41
   RT 45
   FD 60
   RT 45
   FD 30 * 1.41
END
```

```
TO BRAD :X :Y :L
   PU
   SETPOS SENTENCE :X :Y
   PD
   IF :L<2 [STOP]
   FD :L
   RT 120
   FD :L
   BK :L
   LT 60
   BK :L
   FD :L
   LT 60
   BRAD :X :Y+:L :L * 0.6
END

TO BRAZI
   BRAD —100 —60 20
   BRAD —60 —60 15
   BRAD —30 —50 10
END
```

EXEMPLU:

În exemplele prezentate la începutul acestui capitol s-a arătat tehnica de dezvoltare a unui program prin descompunerea în subprograme. Deoarece algoritmul de rezolvare era foarte simplu în ambele cazuri s-au obţinut rezultatele finale dorite cumulînd rezultatele subprogramelor componente. În cazul unor probleme care necesită algoritmi mai complicaţi, în special algoritmi euristici, se poate aborda o altă cale de la general la particular sau de la complex la simplu prin simplificarea iniţială a programelor şi rafinarea succesivă a acestora. Această tehnică va fi exemplificată prin scrierea unui program pentru jocul Nim. Acesta constă din a extrage una, două sau trei beţe dintr-o mulţime dată. Doi jucători execută mutări (extrageri) succesive. Jucătorul care extrage ultimul băţ, adică face ultima mutare acela cîştigă jocul. Iniţial se va scrie o procedură simplă care va fi analizată şi perfecţionată pentru a obţine o serie de programe din ce în ce mai „inteligente".

Astfel primul program nu implementează de fapt o strategie de joc ci pur şi simplu ţine scorul unui joc între două persoane, fără a analiza nici măcar corectitudinea mutărilor acestora. În etapa următoare programul va analiza corectitudinea mutărilor şi va anunţa cîştigătorul. În continuare programul va fi dezvoltat astfel încît să poată el însuşi juca cu programatorul efectuînd mutări aleatoare dar corecte (nu neapărat pe cele mai bune). După ce acest program se execută corect el poate fi rafinat în continuare îmbunătăţindu-i strategia, sau se poate scrie un program complet nou pe baza experienţei acumulate, pînă se obţine un program „inteligent" cu o strategie infailibilă.

Pentru a trece la scrierea programului se vor face următoarele notaţii:

- — JOCNIM numele procedurii care implementează jocul Nim;
- — :BEŢE numărul de beţe rămase neextrase;
- — :JUCĂTOR numele unui jucător;
- — :OPONENT numele partenerului de joc;
- — :MUTARE numărul de beţe ce se vor extrage în această rundă.

JOCNIM trebuie să ţină minte :BEŢE, :JUCĂTOR şi :OPONENT de la runda precedentă şi să asigure preluarea prin :MUTARE a numărului de beţe ce vor fi extrase. Primele trei variabile trebuie transmise de la o rundă la alta deci trebuie să fie intrări. Pe de altă parte :MUTARE este preluată de la tastatură deci nu trebuie să fie intrare. Cînd algoritmul va fi suficient de elaborat ca să execute el însuşi mutări, :MUTARE o să fie generată de o procedură. Preluarea unei mutări de la tastatură va fi realizată cu procedura PRELMUTARE. Iată prima versiune a jocului logic JOCNIM:

```
TO JOCNIM :BEŢE :JUCĂTOR :OPONENT
   PRINT SENTENCE [NUMĂRUL DE BEŢE ESTE] :BEŢE
   PRINT SENTENCE :JUCĂTOR [CARE ESTE MUTAREA TA?]
   MAKE "BETEN :BEŢE — PRELMUTARE
   JOCNIM :BETEN :OPONENT :JUCĂTOR
END

TO CITNUMĂR
   MAKE "IN FIRST READLIST
   IF NUMBERP :IN [OUTPUT :IN] [PRINT [VA ROG INTRODUCEŢI   UN
   NUMĂR] OUTPUT CITNUMĂR
END

TO PRELMUTARE
   MAKE "MUTARE CITNUMĂR
   OUTPUT :MUTARE
END
```

Procedura PRELMUTARE putea fi mai simplă în această etapă, de exemplu fără să testeze dacă răspunsul jucătorului este număr sau altceva. În continuare programul va fi dezvoltat cu următoarele funcţii:

- — detectarea sfîrşitului jocului;
- — declanşarea cîştigătorului la sfîrşitul jocului;
- — verificarea mutărilor jucătorilor dacă sînt 1,2 sau 3 de fiecare dată.

Primele două funcţii sînt implementate astfel:

```
IF :BETEN = O [PRINT SENTENCE :JUCĂTOR [ESTE CÎŞTIGĂTOR!] STOP]
```

A treia funcţie se poate implementa modificînd pe PRELMUTARE astfel ca să efectueze o reîncercare dacă este cazul. Se va utiliza predicatul MEMBERP care primeşte la intrare un element şi o listă şi testează dacă elementul aparţine listei.

```
TO PRELMUTARE
   PRINT  [PUTEŢI  EXTRAGE  1, 2 SAU 3 BEŢE]
   MAKE "MUTARE CITNUMĂR
   IF MEMBER  :MUTARE [1 2 3] [ ] [OUTPUT PRELMUTARE]
   OUTPUT :MUTARE
END
```

Deşi programul este mult mai perfecţionat acum mai are însă multe aspecte nerezolvate. Astfel atunci cînd :BEŢE este 2, PRELMUTARE permite să se extragă 1, 2 sau 3 beţe. Dacă JUCĂTOR extrage 3 beţe, :BEŢE devine negativ şi jocul nu se mai termină niciodată. Rămîne ca exerciţiu eliminarea acestei pene. În această formă programul poate fi completat cu o serie de funcţii noi cum ar fi mutări contra cronometru, declararea cîştigătorului cu una sau două mutări în avans, revenirea la mutarea precedentă, afişarea permanentă a punctajului, îndrumarea jucătorilor privind regulile de joc, schimbarea regulilor de joc, etc.

În continuare se va scrie programul JOCNIM astfel ca să poată efectua propriile sale mutări. Cea mai simplă soluţie pentru efectuarea unei mutări este o alegere aleatoare cu procedura GENMUTARE. Vom obţine astfel:

```
TO PRELMUTARE :JUCĂTOR
   IF :JUCĂTOR = [CALCULATOR] [MAKE "MUTARE GENMUTARE] [PRINT
   ]PUTEŢI EXTRAGE 1, 2 SAU 3 BEŢE] MAKE "MUTARE CITNUMĂR]
   ...................

TO GENMUTARE
   OUTPUT ITEM (1 +RANDOM 3) [1 2 3]
END
```

În această etapă pana dinainte poate avea efecte foarte serioase. O soluţie de eliminare a acestei pene constă în furnizarea numărului de beţe la intrarea procedurii PRELMUTARE:

```
TO PRELMUTARE :JUCĂTOR :BEŢE
; iar după liniile MAKE se introduce textul
IF :MUTARE> :BEŢE [OUTPUT PRELMUTARE :JUCĂTOR :BEŢE]
```

Exersînd programul în această etapă se pot constata următoarele greşeli mari pe care le poate face în timpul jocului:

— mai sînt 5 beţe şi calculatorul extrage două;
— mai sînt şase sau şapte beţe şi calculatorul nu lasă patru;
— mai sînt două sau trei beţe şi calculatorul nu le extrage pe toate!

Pentru a elimina aceste greşeli posibile se va proceda astfel:

— se va testa dacă mai există una, două sau trei beţe:
  IF MEMBERP :BEŢE [1 2 3]...
— dacă testul este adevărat extrage toate beţele:
  OUTPUT :BEŢE
— dacă testul nu este satisfăcut procedează ca mai înainte
  MAKE "MUTARE GENMUTARE

Se obţine astfel procedura:

```
TO EXECUTARE :BEŢE
   IF MEMBERP :BEŢE [1 2 3] [OUTPUT :BEŢE] [OUTPUT GENMUTARE]
END
```

Această procedură se va utiliza în locul procedurii GENMUTARE din PRELMUTARE se mai poate adăuga:

```
   IF :BEŢE = 5 [OUTPUT 1]
```

Pentru a crea un program inteligent, sub forma sa finală, se va proceda astfel:

— jocul se termină cînd un jucător lasă zero beţe după ce a efectuat mutarea sa, deci numărul de beţe rămase după mutare are un rol determinant în stabilirea strategiei;

— pentru a ne asigura că la următoarea mutare vor rămîne zero beţe la mutarea curentă trebuie să lăsăm patru;

— pentru ca la mutarea curentă să lăsăm patru la mutarea precedentă trebuia să lăsăm opt;

— deci pentru a putea să cîştigăm trebuie să lăsăm în urma unei mutări 0, 4, 8, 12, 16... beţe, adică un număr divizibil cu 4;

```
    REMAINDER :NUMĂR 4 să fie 0;
 — mutarea va fi dată astfel de expresia:
    :NUMĂR — REMAINDER :NUMĂR 4
```

Astfel modificînd EXECUTARE se obţine MUTAREINT adică mutare inteligentă cu o strategie invincibilă.

```
TO MUTAREINT :BEŢE
    MAKE "REM REMAINDER :BEŢE 4
    IF :REM = 0 [OUTPUT 1]
    OUTPUT :REM
END
```

Listingul final al programului este:

```
TO JOCNIM :beţe :jucător oponent
PRINT SENTENCE [numărul de beţe este] :beţe
IF NOT (:jucător = [calculator]) [PRINT SENTENCE :jucător [care este mutarea
ta ]]
MAKE "beten :beţe — (PRELMUTARE: jucător :beţe)
IF :beten = 0 [print sentence :jucător [este cîştigător !] STOP]
JOCNIM :beten: oponent :jucător
END

TO PRELMUTARE: jucător :beţe
IF :jucător = [calculator] [make "mutare mutareint] [print [puteţi extrage 1, 2 sau
3 beţe MAKE "mutare CÎTNUMĂR]
IF NOT MEMBERP :mutare [1 2 3] [output prelmutare :jucător :beţe]
IF :mutare >: beţe [output prelmutare :jucător :beţe]
OUTPUT :mutare
END

TO MUTAREINT
MAKE "rem REMAINDER :beţe 4
IF :rem = 0 [output 1]
OUTPUT :rem
END

TO CITNUMĂR
MAKE "in FIRST READLIST
IF NUMBERP :in [output :in] [print [va rog să introduceţi un număr] OUTPUT
CÎTNUMĂR]
END
```

```
JOCNIM 17 [calculator] [Gelu]
numărul de beţe este 17
numărul de beţe este 16
Gelu care este mutarea ta?
puteţi extrage 1, 2 sau 3 beţe
1
numărul de beţe este 15
numărul de beţe este 12
Gelu care este mutarea ta?
puteţi extrage 1, 2 sau 3 beţe
6
```

puteţi extrage 1, 2 sau 3 beţe
2
numărul de beţe este 10
numărul de beţe este 8
Gelu care este mutarea ta?
puteţi extrage 1, 2 sau 3 beţe
2
vă rog să introduceţi un număr
1
numărul de beţe este 7
numărul de beţe este 4
Gelu care este mutarea ta?
puteţi extrage 1, 2 sau 3 beţe
2
numărul de beţe este 2
calculator este cîştigător
?

În exemplele prezentate s-au utilizat proceduri recursive.

Pentru a înţelege bine exemplele trebuie parcurs cu atenţie paragraful referitor la recursivitate.

12.2. Recursivitatea în LOGO

Recursivitatea constituie o modalitate de a scrie proceduri care se apelează pe ele înseşi.

Pentru a înţelege mai bine procesul de recursivitate să reamintim ce se întîmplă în momentul în care o procedură apelează o altă procedură, (figura 12.3):

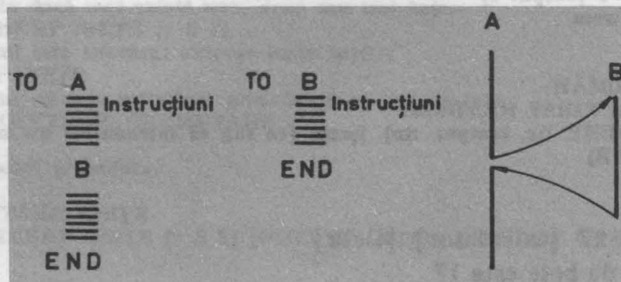


Fig. 12.3. Apelul unei proceduri

Trebuie reamintit că fiecare procedură are un set de date proprii (private) care în general nu sînt asociate ei ci apelului procedurii respective.

În momentul execuţiei procedurii A, se ajunge la apelul procedurii B, moment în care au loc următoarele operaţii, (figura 12.4).

— suspendarea temporară a execuţiei procedurii A şi salvarea contextului (valorile variabilelor);

— activarea setului de date proprii procedurii B, asociat apelului respectiv.

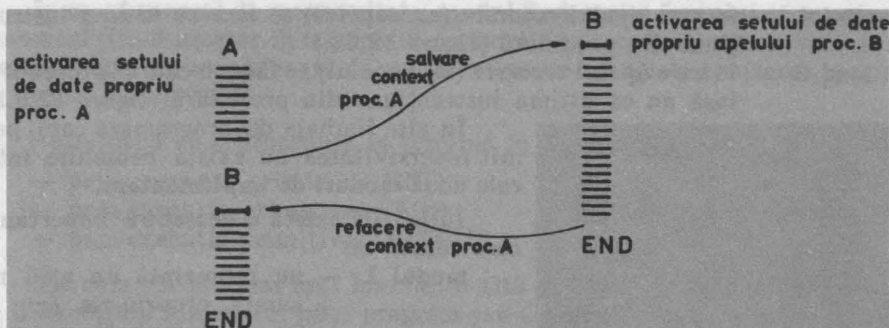


Fig. 12.4. Operaţiile de bază într-un apel de procedură

Se continuă cu execuţia procedurii B şi în momentul terminării acesteia se desfăşoară următoarele operaţii:

— se reface contextul procedurii A (context care a fost salvat în momentul trecerii la procedura B);

— revenirea în procedura A la prima instrucţiune de după apelul procedurii B.

Salvarea şi refacerea contextului se face în mod implicit, de către limbajul LOGO, fără specificarea explicită de către utilizator.

Limbajul LOGO, faţă de alte limbaje de programare cum ar fi BASIC, FORTRAN etc., asigură mecanismul ca o procedură să se apeleze pe ea însăşi, altfel spus are implementat mecanismul de recursivitate.

Avînd în vedere că o procedură se poate chema pe ea însăşi, prin apel recursiv, rezultă faptul că setul de date proprii nu trebuie să fie asociat producerii ci apelului procedurii respective.

La fiecare apel al procedurii respective, se activează un set de date proprii acelui apel. Rutina care se cheamă pe ea însăşi, coexistă cu apelurile anterioare de date proprii asupra cărora lucrează (figura 12.5).

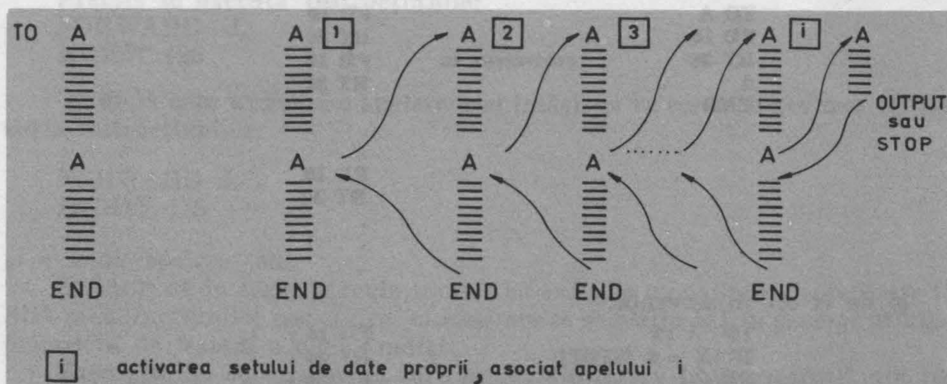


Fig. 12.5. Coexistenţa unei proceduri în apelul recursiv

În limbajul LOGO sînt implementate două tipuri de apeluri recursive şi anume:

mod 1 — în care apelul recursiv (autoapelul) se face ca ultima instrucţiune din procedură (figura 12.6.a)

mod 2 — în care apelul recursiv (autoapelul) se face în cadrul procedurii, însă nu ca ultima instrucţiune din procedură (figura 12.6.b).

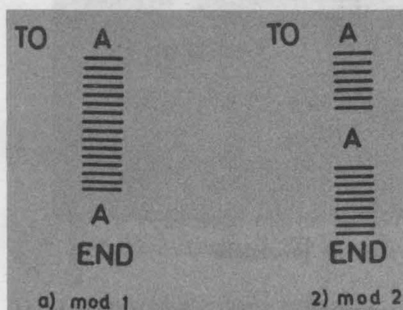


Fig. 12.6. Tipuri de implementare a apelurilor recursive

În alte limbaje de programare care permit recursivitatea nu există deosebire între cele două moduri de implementare.

În LOGO există o deosebire importantă care constă în:

modul 1 — nu reprezintă un apel recursiv propriu-zis (nu se face salvare/refacere context) ci se lucrează pe aceleaşi set de date proprii.

modul 2 — reprezintă un apel recursiv cu salvare context, activare set de date proprii apelului curent, refacere context.

Modul 1 este mai degrabă o extensie a primitivei REPEAT, fiind asemănătoare cu un salt necondiţionat la prima instrucţiune din procedură.

Grupul de instrucţiuni: A
END

apel de procedură şi END sînt tratate împreună şi echivalate cu transferul controlului la prima instrucţiune din procedură, fără a se mai face salvare context, activare set de date proprii apelului, refacere context, ci numai actualizarea setului de date proprii.

În acest fel se poate repeta un grup de instrucţiuni, de un număr de ori nespecificat la primul apel de procedură, fără a avea constrîngeri din punct de vedere al memoriei.

EXEMPLU:

Grupul de instrucţiuni FD 10 RT 30 se repetă la infinit în secvenţa:

```
TO A                              FD 10
FD 10                             RT 30
RT 30          echivalent cu      FD 10
A                                 RT 30
END                               .
                                  .
                                  .
                                  FD 10
                                  RT 30
                                  .
                                  .
                                  .
```

şi de N ori în secvenţa:

```
TO A :N                           FD 10 :  }1
IF :N = 0 [STOP]                  RT 30 :  }
FD 10                             FD 10 :  }2
RT 30          echivalent cu      RT 30 :  }
A :N—1                            .
END                               .
                                  FD 10 :  }N
                                  RT 30 :  }
```

În cazul general de recursivitate, modul 2, datorită faptului că se salvează contextul (fiind necesar în momentul revenirii în procedură) este nevoie de memorie şi după un număr mare de apeluri se poate ajunge la epuizarea spaţiului de memorie disponibil.

Reamintim că o procedură se termină în trei moduri:

— prin execuţia primitivei END
— prin execuţia primitivei STOP
— prin execuţia primitivei OUTPUT

Aşa cum am precizat mai înainte, o altă posibilitate de a repeta un grup de instrucţiuni din cadrul unui program sau a unei proceduri, decît utilizarea primitivei REPEAT, este aceea de a include în procedura respectivă o apelare a ei însăşi.

EXEMPLU:

```
TO TRIUNGHI :L
FORWARD :L
RIGHT 120
TRIUNGHI :L
END
```

— va desena un triunghi cu latura L şi va repeta mereu desenarea acestuia pînă cînd se întrerupe programul cu BREAK.
Deci procedura de mai sus este echivalentă cu:

```
FORWARD :L
RIGHT 120
FORWARD :L
RIGHT 120
FORWARD :L
RIGHT 120
FORWARD :L
RIGHT 120
.
.
.
```

Practic se execută instrucţiunile:
FORWARD :L
RIGHT 120

— după care urmează o apelare a ei însăşi, ce va conduce din nou la execuţia instrucţiunilor:

FORWARD :L
RIGHT 120

şi o nouă apelare etc.

Evident că în cazurile reale trebuie să existe o modalitate de repetare finită a instrucţiunilor respective, modalitate ce se realizează în general printr-o primitivă de testare a unei condiţii.

Exemplu: Să construim un turn ca cel din figura 12.7, construit din pătrate a căror latură se micşorează continuu.

Operaţia de construire a turnului este un proces repetitiv. Să presupunem că nu ne interesează numărul de piese puse una peste alta, ci următoarele caracteristici:

— latura piesei de bază L_1 =valoarea specificată de utilizator;
— relaţia între latura piesei următoare şi a celei curente:
$L_{i+1}=0.5 * L_i$ ($L_2=0.5 * L_1$; $L_3=0.5 * L_2$, etc.);
— ultima piesă să nu aibă latură mai mică de 2 puncte $L_n > 2$.

Construirea turnului se va face printr-o procedură recursivă (se va apela pe ea însăşi) şi de fiecare dată se va modifica parametrul de apelare şi totodată se va testa dacă latura L_n a devenit mai mică decît 2. În caz afirmativ se va încheia procesul de desenare.

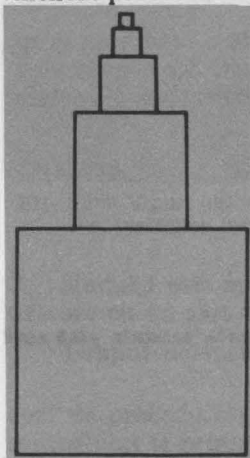


Fig. 12.7. TURN construit prin recursivitate

```
TO TURN :L
IF :L < 2 [STOP]
PĂTRAT :L
FORWARD :L
RIGHT 90
FORWARD :L * 0.25
LEFT 90
TURN :L * 0.5
END
```

Corpul procedurii va genera un desen de bază ca cel din figura 12.8, care va constitui piesa de bază a procesului de repetare.

În cadrul procedurii se apelează o procedură de construire a pătratului.

```
TO PĂTRAT :L
REPEAT 4 [FORWARD :L RIGHT 90]
END
```

— după care se mută penelul în poziţia necesară construirii următoarei piese a turnului.

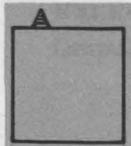


Fig. 12.8. Piesa de bază în construirea repetitivă a turnului

Construirea turnului se realizează apelînd procedura TURN 32

Procesul se repetă prin apelarea recursivă a procedurii TURN cu altă valoare a laturii piesei şi de fiecare dată în cadrul procedurii se verifică dacă latura nu a devenit mai mică decît 2.

Ambele proceduri TRIUNGHI, TURN au fost realizate în primul mod de implementare (echivalente cu repetarea grupului de instrucţiuni).

EXEMPLU: Pentru a înţelege mai bine recursivitatea şi pentru a scoate în evidenţă diferenţa dintre modul 1 şi modul 2 de implementare să analizăm două proceduri care din punct de vedere al conţinutului par asemănătoare însă din punct de vedere al execuţiei produc rezultate complet diferite:

```
TO NUMĂRĂ.INV :NR
IF :NR = 0 [STOP]
PRINT :NR
NUMĂRĂ.INV :NR—1
END
```

```
TO NUMĂRĂ.DIR :NR
IF :NR = 0 [STOP]
NUMĂRĂ.DIR :NR—1
PRINT :NR
END
```

Apelul acestor proceduri va genera următoarele rezultate:

NUMĂRĂ.DIR 4

1
2
3
4

NUMĂRĂ.INV 4

4
3
2
1

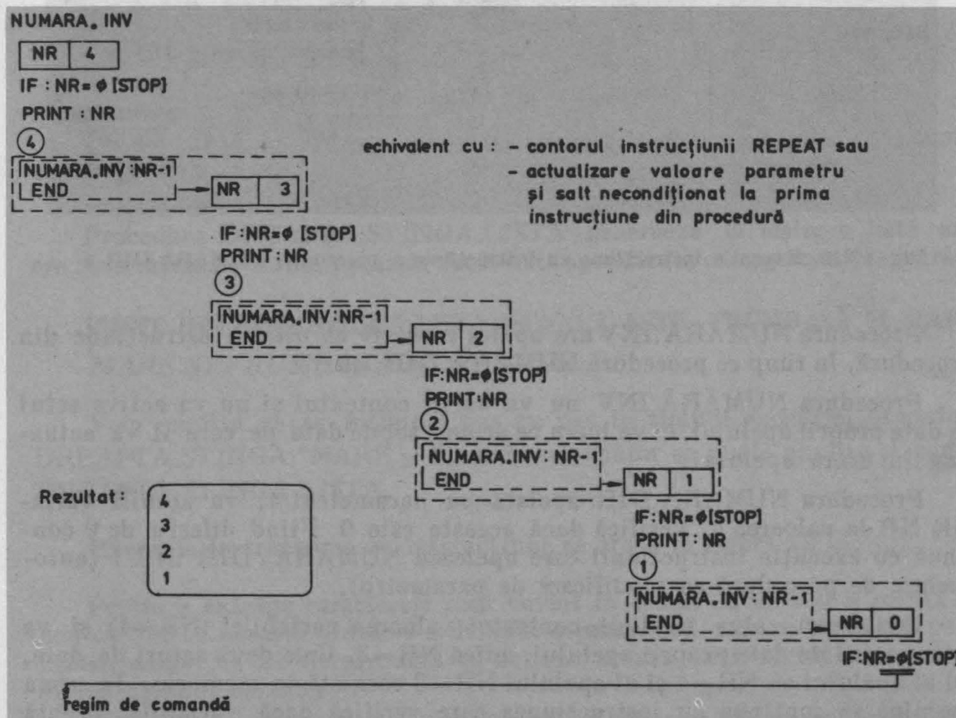


Fig. 12.9. Execuţia instrucţiune cu instrucţiune a procedurii NUMARA.INV 4

Să analizăm procesul de execuţie a primitivelor în ambele situaţii.

Execuţia instrucţiune cu instrucţiune a procedurii NUMĂRĂ.INV 4 este prezentă în figura 12.9, iar a procedurii NUMĂRĂ.DIR 4 este prezentată în figura 12.10. Se recomandă ca parametrul NR să primească valori naturale în momentul apelării procedurilor.

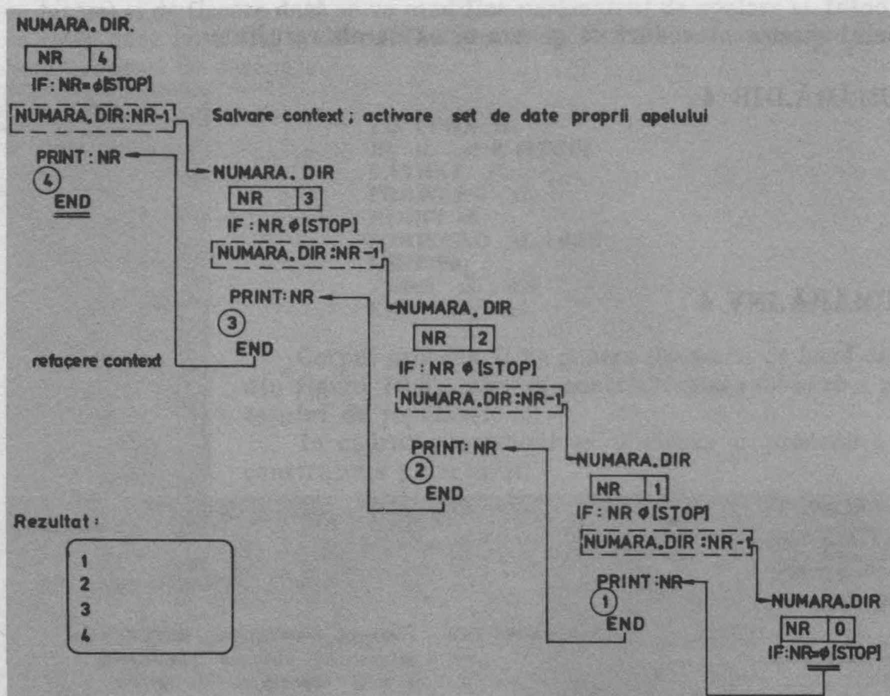


Fig. 12.10. Execuţia instrucţiune cu instrucţiune a procedurii NUMARA.DIR 4

Procedura NUMĂRĂ.INV are apelul recursiv ca ultimă instrucţiune din procedură, în timp ce procedura NUMĂRĂ.DIR, nu.

Procedura NUMĂRĂ.INV nu va salva contextul şi nu va activa setul de date proprii apelului, ci va lucra pe acelaşi set de date pe care îl va actualiza în urma apelului.

Procedura NUMĂRĂ.DIR apelată cu parametrul 4, va stabili variabila NR la valoarea 4. Verifică dacă aceasta este 0. Fiind diferită de 0 continuă cu execuţia instrucţiunii care apelează NUMĂRĂ.DIR :NR-1 (autoapelare de procedură cu modificare de parametru).

LOGO va salva implicit contextul (valoarea variabilei :NR=4) şi va activa setul de date proprii apelului, adică NR=3. Cele două seturi de date, cel al apelului cu NR=4 şi al apelului NR=3 coexistă în memorie. În urma apelului se continuă cu instrucţiunea care verifică dacă variabila curentă (proprie apelului) a ajuns la zero. În caz contrar se continuă apelul recursiv pînă cînd variabila curentă ajunge la 0. În acel moment se execută instruc-

ţiunea STOP care reprezintă terminarea apelului curent al procedurii, se reface implicit contextul şi se revine în procedura chematoare. Se continuă cu instrucţiunea PRINT :NR, care va produce afişarea valorii 1.

Întîlnind instrucţiunea END se reface contextul şi se revine în procedura chemătoare, etc.

În final se revine în regim de comandă.

EXEMPLU:

Recursivitatea se poate utiliza în mod eficient în proceduri care efectuează aceleaşi operaţii pentru toate elementele unui obiect. Vom exemplifica utilizarea recursivităţii cu două proceduri care scriu invers (de la dreapta la stînga) caracterele unui cuvînt sau cuvintele dintr-o propoziţie (în general elementele unei liste).

```
TO DREAPTA.STÎNGA :C
IF EMPTYP :C [OUTPUT " ]
OUTPUT WORD (LAST :C) (DREAPTA.STÎNGA BUTLAST :C)
END
```

Procedura DREAPTA.STÎNGA generează la ieşire un cuvînt care are caracterele în ordinea inversă decît ale cuvîntului specificat ca argument de intrare.

PRINT DREAPTA.STINGA "MARE
ERAM

```
TO DREAPTA "STÎNGA.LISTA :L
IF EMPTYP :L [OUTPUT :L]
OUTPUT SENTENCE (LAST :L) (DREAPTA.STÎNGA.LISTA BUTLAST:L)
END
```

Procedura DREAPTA.STÎNGA.LISTA generează la ieşire o listă care are elementele în ordine inversă, decît lista specificată ca argument de intrare.

PRINT DREAPTA.STINGA.LISTA [ŞCOALA ESTE FRUMOASĂ ŞI MARE]
MARE ŞI FRUMOASĂ ESTE ŞCOALA

Vom analiza ce se întîmplă pas cu pas în timpul execuţiei procedurii DREAPTA.STÎNGA "MARE şi în mod asemănător se poate analiza şi pentru DREAPTA.STÎNGA.LISTA.

Execuţia instrucţiune cu instrucţiune se arată în figura 12.11.

Pentru a extrage caracterele unui cuvînt în ordine inversă şi a realiza un nou cuvînt cu această ordine s-a utilizat o procedură care se apelează pe ea însăşi, numai că de fiecare dată va acţiona asupra unui cuvînt din care s-a extras ultimul caracter.

Procesul recursiv continuă pînă cînd se epuizează toate caracterele din cuvînt.

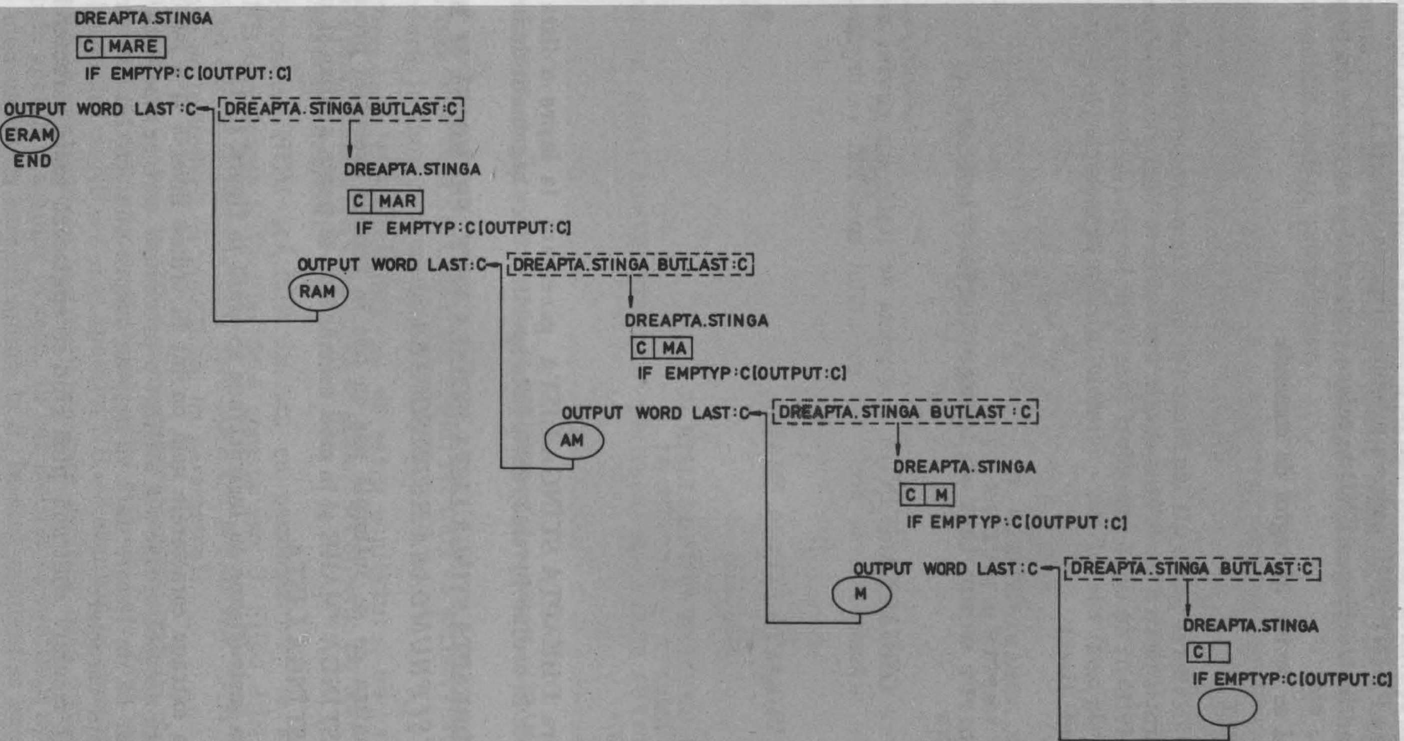


Fig. 12.11. Execuţia instrucţiune cu instrucţiune a procedurii DREAPTA. STÎNGA

EXEMPLU:

Un alt exemplu de utilizare a recursivității este acela în care se desenează un arbore ca cel din figura 12.12.

Dacă analizăm atent, se observă că în fiecare nod putem considera că se construiește un nou arbore ale cărui ramuri au dimensiuni din ce în ce mai mici.

În nodurile extreme arborele se reduce la un arbore elementar format numai din două ramuri (figura 12.13).

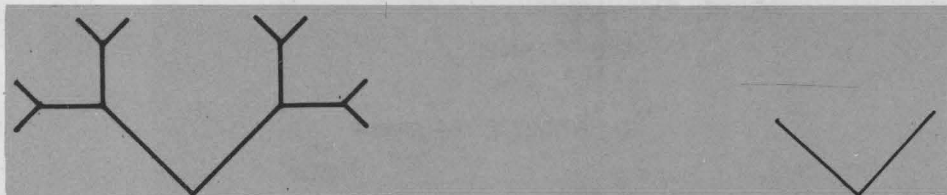


Fig. 12.12. ARBORE desenat prin recursivitate

Fig. 12.13. ARBORE elementar

Este evident faptul că pentru a construi arborele total se apelează recursiv în fiecare nod construirea unui arbore cu dimensiuni reduse.

Primitivele pentru construirea arborelui elementar sînt:

```
LEFT 45          ; desenarea ramurii stînga
FORWARD :L
----------
BACK    :L
RIGHT 90         ;revenire şi desenare ramura dreapta
FORWARD :L
----------
BACK    :L
LEFT 45          ;revenire în poziţia iniţială
```

O precizare importantă care trebuie făcută este aceea că în procesul de recursivitate este necesar să se stabilească precis starea inițială a penelului.

În descrierea arborelui elementar s-a revenit în aceeași poziție și cu aceeași direcție ca în momentul începerii lui.

În procesul construirii arborelui total, desenăm mai întîi o ramură stînga a arborelui elementar, după care am ajuns într-un nod în care trebuie să considerăm că pornim procesul de construire a unui nou arbore.

Deci după:

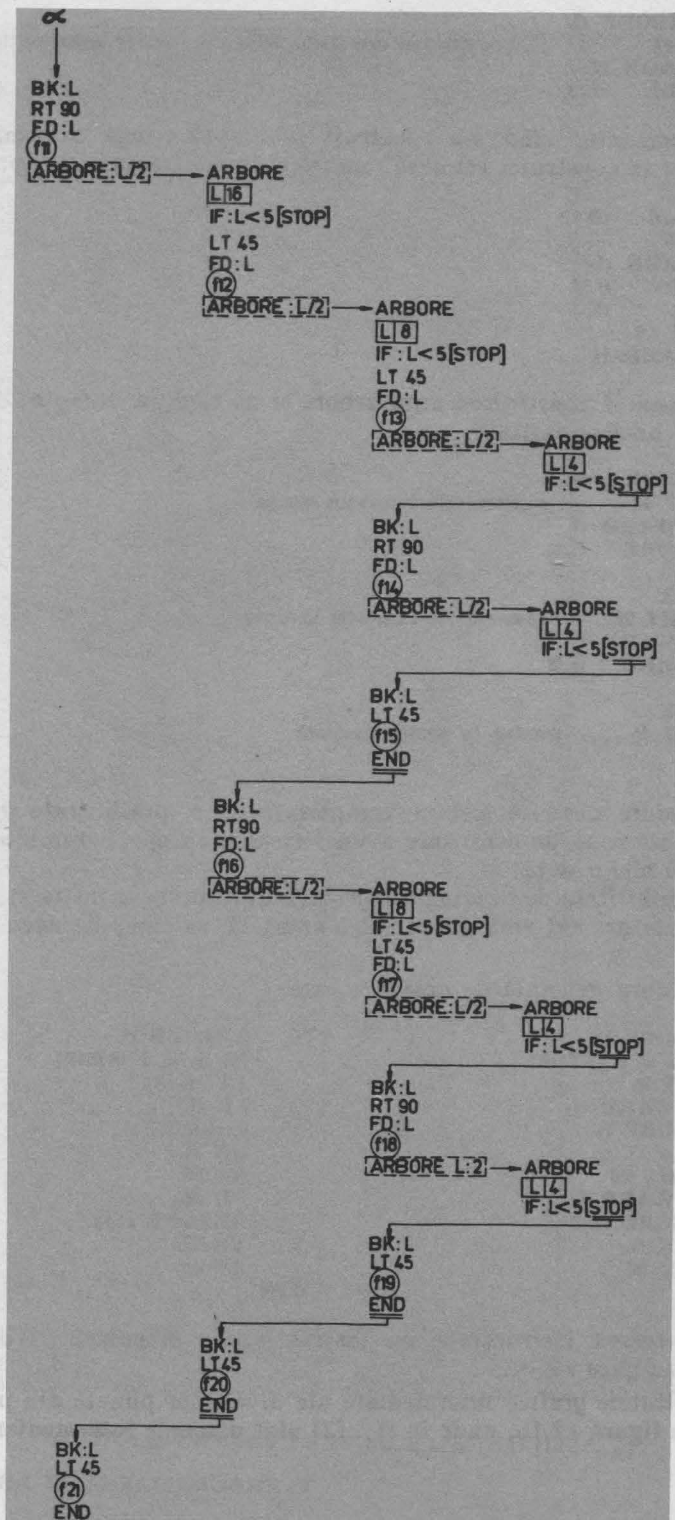
```
TO ARBORE: L
LEFT 45         ;prima ramură stînga a arborelui
FORWARD :L
```

— trebuie să apelăm procedura ARBORE de construire a unui nou arbore cu un parametru de latură modificat.

Vom considera că un subarbore are ramura inițială jumătate cît cea a arborelui pe nivelul anterior.

```
ARBORE
L 32
IF :L<5 [STOP]
LT 45
FD :L
(f1)
ARBORE :L/2 ——→ ARBORE
                L 16
                IF :L<5 [STOP]
                LT 45
                FD :L
                (f2)
                ARBORE :L/2 ——→ ARBORE
                                L 8
                                IF :L<5 [STOP]
                                LT45
                                FD :L
                                (f3)
                                ARBORE :L/2 ——→ ARBORE
                                                L 4
                                                IF :L<5 [STOP]
                                BK :L
                                RT 90
                                FD :L
                                (f4)
                                ARBORE :L/2 ——→ ARBORE
                                                L 4
                                                IF :L<5 [STOP]
                                BK :L
                                LT 45
                                (f5)
                                END
                BK :L
                RT 90
                FD :L
                (f6)
                ARBORE :L/2 ——→ ARBORE
                                L 8
                                IF :L<5 STOP
                                LT 45
                                FD :L
                                (f7)
                                ARBORE :L/2 ——→ ARBORE
                                                L 4
                                                IF :L<5 [STOP]
                                BK :L
                                RT 90
                                FD :L
                                (f8)
                                ARBORE :L/2 ——→ ARBORE
                                                L 4
                                                IF :L<5 [STOP]
                                BK :L
                                LT 45
                                (f9)
                                END
                BK :L
                LT 45
                (f10)
                END
α
```

Fig. 12.14. Execuţia instrucţiune cu instrucţiune a procedurii ARBORE


α
BK:L
RT 90
FD:L
f11
ARBORE :L/2
ARBORE
L 16
IF:L< 5[STOP]
LT 45
FD:L
f12
ARBORE :L/2
ARBORE
L 8
IF : L< 5[STOP]
LT 45
FD:L
f13
ARBORE :L/2
ARBORE
L 4
IF:L<5[STOP]
BK:L
RT 90
FD:L
f14
ARBORE :L/2
ARBORE
L 4
IF:L< 5[STOP]
BK:L
LT 45
f15
END
BK:L
RT90
FD:L
f16
ARBORE:L/2
ARBORE
L 8
IF:L<5[STOP]
LT 45
FD:L
f17
ARBORE:L/2
ARBORE
L 4
IF: L<5[STOP]
BK:L
RT 90
FD:L
f18
ARBORE L:2
ARBORE
L 4
IF:L<5[STOP]
BK:L
LT 45
f19
END
BK:L
LT45
f20
END
BK:L
LT 45
f21
END

```
TO ARBORE :L
LEFT 45              ;construirea ramurilor stinga a tuturor subarborilor
FORWARD :L
ARBORE  :L/2
```

În momentul cînd s-a construit o ramură stînga extremă trebuie să revenim şi să construim cealaltă ramură extremă (ramura dreapta).

```
TO ARBORE   :L
   LEFT 45
   FORWARD :L
   ARBORE  :L/2
   BACK    :L
   RIGHT 90
   FORWARD :L
```

Se încearcă construirea unui arbore şi pe ramura dreapta, după care revenim în poziţia iniţială.

```
TO   ARBORE   :L
     LEFT 45      ; construieşte ramurile stînga
     FORWARD :L
     ARBORE  :L/2
     ..............
     BACK    :L
     RIGHT 90     ; construieşte ramurile dreapta
     FORWARD :L
     ARBORE  :L/2
     ................
     BACK    :L
     LEFT 45      ;revine în poziţia iniţială
END
```

Procedura descrisă trebuie completată cu o posibilitate de terminare, deoarece procesul de construire a unei ramuri în speţă ramura stînga) nu se termină nici o dată.

Ca posibilitate de terminare a procesului recursiv se poate stabili momentul în care latura extremă ajunge sub o anumită valoare, de exemplu valoarea 5.

Procedura generală de desenare este:

```
TO   ARBORE :L                 TO   ARBORE :L
     IF :L < 5 [STOP]               IF :L < 5 [STOP]
     LEFT 45                        LT 45
     FORWARD :L                     FD :L
     ARBORE :L/2          sau       ARBORE :L/2
     BACK :L                        BK :L
     RIGHT 90                       RT 90
     FORWARD L:                     FD :L
     ARBORE :L/2                    ARBORE :L/2
     BACK :L                        BK :L
     LEFT 45                        LT 45
END                            END
```

Parcurgerea instrucţiune cu instrucţiune a procedurii ARBORE 32 este arătată în figura 12.14.

Rezultatele grafice intermediare ale diverselor puncte din procedură sînt arătate în figura 12.15, unde în f1...f21 sînt desenele intermediare.

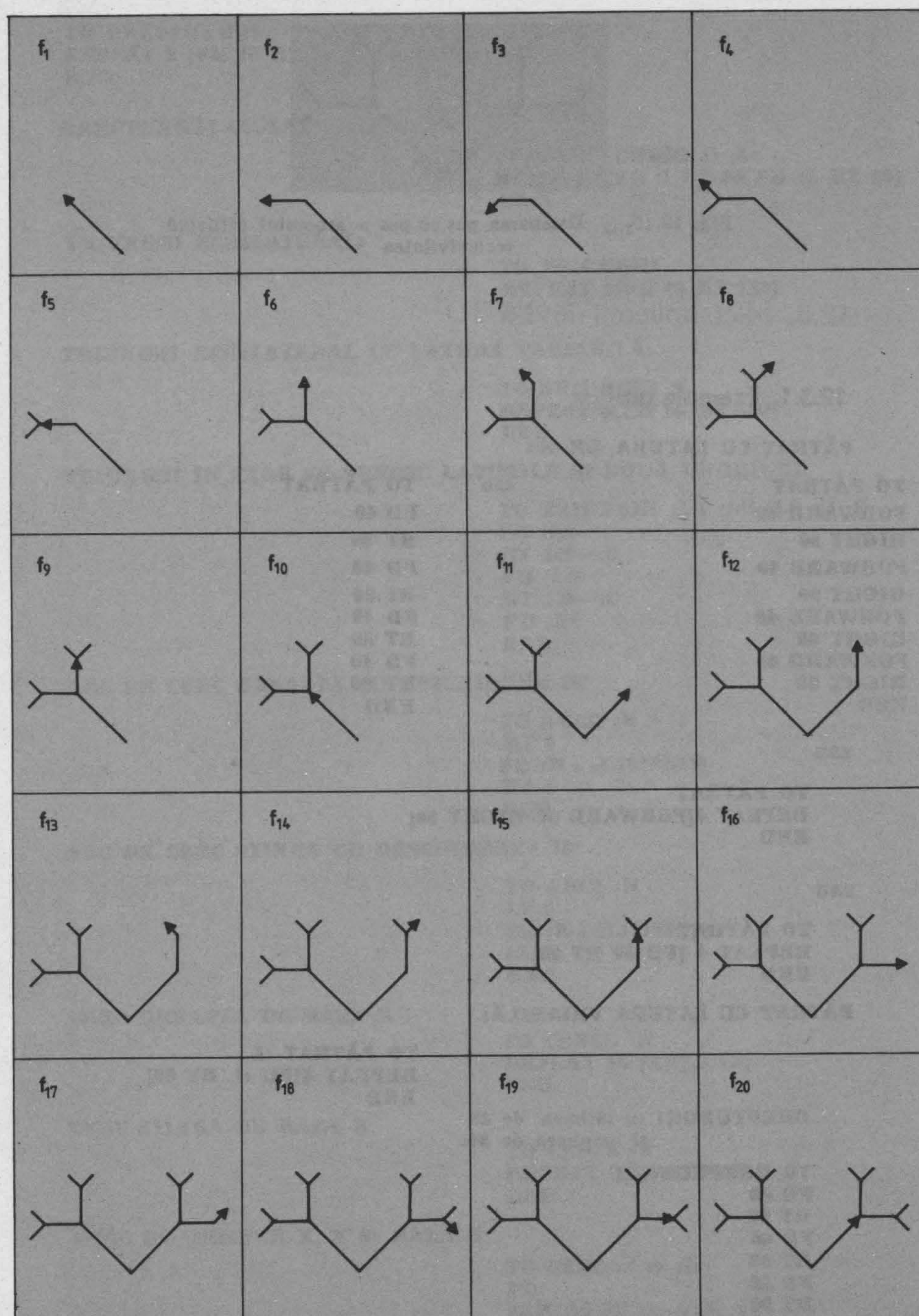


Fig. $12.15_{1..20}$ Desenarea pas cu pas a arborelui ($12.15_{1..20}$)

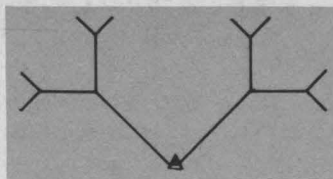


Fig. $12.15_{1.21}$ Desenarea pas cu pas a arborelui utilizînd recursivitatea

12.3. Mici aplicaţii în LOGO

12.3.1. Exemple grafice

PĂTRAT CU LATURA DE 40:

```
TO PĂTRAT
FORWARD 40
RIGHT 90
FORWARD 40
RIGHT 90
FORWARD 40
RIGHT 90
FORWARD 40
RIGHT 90
END
```

sau

```
TO PĂTRAT
FD 40
RT 90
FD 40
RT 90
FD 40
RT 90
FD 40
RT 90
END
```

sau

```
TO PĂTRAT
REPEAT 4[FORWARD 40 RIGHT 90]
END
```

sau

```
TO PĂTRAT
REPEAT 4 [FD 40 RT 90]
END
```

PĂTRAT CU LATURA VRIABILĂ:

```
TO PĂTRAT :L
REPEAT 4[FD :L RT 90]
END
```

DREPTUNGHI cu lăţimea de 40 şi lungimea de 90:

```
TO DREPTUNGHI
FD 40
RT 90
FD 60
RT 90
FD 40
RT 90
FD 60
RT 90
END
```

```
TO DREPTUNGHI
REPEÁT 2 [FD 40 RT 90 FD 60 RT 90]
END
```

DREPTUNGHI CU LATURILE VARIABILE:

```
TO DREPTUNGHI :I :L
REPEAT 2[FD :I RT 90 FD :L RT 90]
END
```

TRIUNGHI ECHILATERAL:

```
TO TRIUNGHI
REPEAT 3[FD 40 RT 120]
END
```

TRIUNGHI ECHILATERAL CU LATURA VARIABILĂ:

```
TO TRIUNGHI :L
REPEAT 3[FD :L RT 120]
END
```

TRIUNGHI ÎN CARE SE CUNOSC LATURILE ŞI DOUĂ UNGHIURI:

```
TO TRIUNGHI :LA :LB :LC :B :C
FD :LA
RT 180—:B
FD :LB
RT 180—:C
FD :LC
END
```

ARC DE CERC DREAPTA CU DESCHIDERE 10°

```
TO ARCD :R
RT 5
FD :R * (3.14159/18)
RT 5
END
```

ARC DE CERC STÎNGA CU DESCHIDEREA 10°

```
TO ARCS :R
LT 5
FD :R * (3.14159/18)
LT 5
END
```

CERC DREAPTA DE RAZA R

```
TO CERCD :R
REPEAT 36 [ARCD :R]
END
```

CERC STÎNGA DE RAZA R

```
TO CERCS :R
REPEAT 36 [ARCS :R]
END
```

CERC DE ORIGINE X, Y ŞI RAZĂ R

```
TO CERC :x :y :R
PU
SEPTOS SE :x — :R :y
PD
CERCD :R
END
```

```
sau
TO CERC :x :y :R
PU
SEPTOS SE :x + :R :y
PD
CERCS :R
END
```

EXEMPLU DE UTILIZARE ARCD ŞI ARCS
RAZA DE SOARE

```
TO RAZAS :L
REPEAT 2[ARCD :L ARCS :L]
END
```

SOARE

```
TO SOARE :L
REPEAT 9[RAZAS :L RT 160]
END
```

Se recomandă SOARE 18

POLIGON DE LATURA L ŞI UNGHI U

```
TO POLIGON :L :U
FD :L
RT :U
POLIGON :L :U
END
```

De remarcat că desenul nu se termină nici o dată.
Trebuie oprit cu break.
Se obţin figuri interesante pentru următoarele valori:

L	U
50	160
60	80
80	144
20	40
80	156
60	90
50	60

SPIRALA

```
TO SPIRALA :L :U :INC
FD :L
RT :U
SPIRALA (:L + :INC) :U :INC
END
```

De remarcat că programul nu se termină decît cu break
Figurile interesante se obţin pentru:

L	U	INC
1	75	1
1	45	1
1	45	3
5	120	3
1	144	3
1	150	1
1	88	1

```
BRAD
                    TO RAMURA :L
                    FD :L
                    RT 120
                    FD :L
                    BK :L
                    LT 60
                    BK :L
                    FD :L
                    LT 60
                    END

                    TO BRAD :L
                    IF :L < 2 [STOP]
                    RAMURA :L
                    BRAD :L * 0.6
                    END

                    BRAD 30
```

12.3.2. Suma a două numere aleatoare

Vom scrie un program simplu care permite verificarea cunoştinţelor despre operaţiile aritmetice elementare.

Programul va genera două numere aleatoare între 0 şi 1000 şi va întreba utilizatorul cît este suma acestora. În funcţie de răspunsul primit va preciza dacă acesta este corect sau nu şi în cazul în care răspunsul este incorect va specifica răspunsul bun.

Procesul de întrebare / răspuns va continua pînă cînd utilizatorul va întrerupe execuţia programului cu BREAK.

```
TO SUMA
MAKE   "NUM1  RANDOM  1000
MAKE   "NUM2  RANDOM  1000
MAKE   "REZ   :NUM1   +  :NUM2
PRINT  (SENTENCE  [CIT FACE] :NUM1 [+]  : NUM2)
MAKE   "RASP READLIST
IF     :REZ = FIRST :RASP [PRINT[ RĂSPUNS CORECT ]SUMA]
PRINT SENTENCE [RASPUNSUL CORECT ESTE] :REZ
SUMA
END
```

În primele două instrucţiuni se atribuie variabilelor NUM1 şi NUM2 cîte un număr, aleator, cuprins între 0 şi 1000.

Suma acestor două numere, care constituie rezultatul se atribuie variabilei REZ.

Se constituie o listă formată din elementele
- lista [CÎT FACE]
- variabila NUM1
- lista [+]
- variabila NUM2

care se tipăreşte, reprezentînd întrebarea la care trebuie să răspundă utilizatorul.

Răspunsul utilizatorului se atribuie variabilei RĂSP.

Se verifică dacă răspunsul RĂSP (care fiind sub forma de listă trebuie să-i extragem primul element FIRST :RĂSP) este egal cu rezultatul REZ. Dacă da, se tipăreşte mesajul răspuns corect şi se continuă. În caz contrar, se indică care este răspunsul corect şi se continuă.

Trebuie menţionat faptul că acest program simplu nu este protejat la introducerea unor răspunsuri care nu sînt numere.

Pentru acesta este necesar să se introducă o procedură care verifică acest lucru.

```
TO CITNR
MAKE "NR FIRST READLIST
IF NUMBERP :NR [OUTPUT :NR]
PRINT [VA ROG INTRODUCEŢI NUMĂR]
OUTPUT CITNR
END
```

Procedura verifică dacă răspunsul este număr.

În caz afirmativ, generează ca ieşire numărul respectiv iar în caz contrar anunţă faptul că trebuie introdus, ca răspuns, un număr şi aşteaptă introducerea acestuia.

Programul se modifică în felul următor:

```
TO SUMA
MAKE "NUM1 RANDOM 1000
MAKE "NUM2 RANDOM 1000
MAKE "REZ :NUM1 + :NUM2
PRIN (SENTENCE [CIT FACE] :NUM1 [+] :NUM2)
MAKE "RASP CITNR
IF :REZ = :RASP [PRINT [RĂSPUNS CORECT] SUMA]
PRINT SENTENCE [RĂSPUNSUL CORECT ESTE] :REZ
SUMA
END
```

Acest program simplu poate fi extins. Ca exerciţiu pentru cititor sugerăm completarea programului cu următoarele facilităţi:

— dacă utilizatorul greşeşte răspunsul, programul să nu indice imediat care este rezultatul, ci să întrebe de 2—3 ori pe utilizator şi numai după aceea să-i indice răspunsul corect.

— să se alcătuiască un punctaj al răspunsurilor corecte/incorecte şi să se configureze pentru diverse operaţii +, —, *, /.

— dacă utilizatorul răspunde corect, întrebările să nu mai fie simple, ci să le transforme în expresii din ce în ce mai complicate.

12.3.3. Ordonare alfabetică

Să se scrie un program LOGO care are ca intrare o listă de elemente într-o ordine oarecare şi generează la ieşire o altă listă ordonată alfabetic

Principiul de sortare constă în extragerea element cu element din lista sursă (iniţială) şi transferarea lor în lista destinaţie (finală). În timpul transferării elementul curent este inserat în lista destinaţie astfel încît aceasta să fie ordonată alfabetic.

Programul poate fi utilizat şi pentru a concatena două liste, iar rezultatul să fie ordonat alfabetic, cu condiţia ca una din liste să fie deja ordonată.

În procesul de transferare este necesară o procedură de inserare a unui element în lista destinaţie.

```
TO INSERARE     :ELEM     :LISTA
IF      EMPTYP  :LISTA [ OUTPUT LIST :ELEM " ]
IF      ORDONAT :ELEM FIRST :LISTA [ OUTPUT FPUT :ELEM :LISTA]
OUTPUT FPUT FIRST :LISTA INSERARE :ELEM BUTFIRST :LISTA
END
```

Dacă lista destinaţie LISTA este vidă, atunci se creează o listă cu elementul ELEM pe care dorim să-l introducem. În caz contrar verificăm dacă elementul îşi are locul înaintea tuturor elementelor din lista destinaţie. Dacă da, îl introducem în listă. Dacă nu, verificăm pe rînd dacă îşi are locul după primul, al doilea, etc. element din lista destinaţie, apelînd recursiv procesul de înserare aplicat unei liste din care eliminăm rînd pe rînd elementele care trebuie să fie înaintea elementului ELEM specificat.

Procedura ORDONAT generează valoarea logică TRUE sau FALSE dacă primul argument este înaintea celui de-al doilea argument în ceea ce priveşte ordinea alfabetică. Compararea se face pe baza codurilor ASCII.

```
TO    ORDONAT :ARG1    :ARG2
IF    :ARG1 = "    [OUTPUT TRUE ]
IF    :ARG2 = "    [ OUTPUT FALSE ]
IF    (ASCII FIRST  :ARG1) < (ASCII FIRST  :ARG2) [ OUTPUT TRUE ]
IF    (ASCII FIRST  :ARG1) > (ASCII FIRST  :ARG2) [OUTPUT FALSE ]
OUTPUT ORDONAT BUTFIRST  :ARG1 BUTFIRST  :ARG2
END
```

Procedura de sortare SORTARE se poate scrie astfel:

```
TO SORTARE:  :LISTAI :LISTAF
IF EMPTYP     :LISTAI [OUTPUT   :LISTAF]
MAKE "LISTAF INSERARE FIRST    :LISTAI :LISTAF
OUTPUT  SORTARE  BUTFIRST   :LISTAI :LISTAF
END
```

Exemplu de utilizare:

```
MAKE "LISTA.SORTATA SORTARE [M C A B F] [ ]
PRINT :LISTA.SORTATA
A B C F M
MAKE "LISTA. SORTATA SORTARE [ELEV BINE NOTA] :LISTA.SORTATA
PRINT :LISTA.SORTATA
A B BINE C ELEV F NOTA M
```

12.3.4. Conversia numerelor naturale din baza 10 într-o bază <16

Să se scrie un program care realizează conversia numărului natural N din baza 10 într-o bază B<16.

Se ştie că un număr în baza B foloseşte pentru reprezentare cifrele utilizate în baza respectivă. Astfel, în baza 16 se utilizează:

0 1 2 3 4 5 6 7 8 9 A B C D E F

iar într-o bază < 16 se utilizează un subset al acestora.

Conversia unui număr întreg se realizează prin concatenarea resturilor împărţirii numărului la baza în care dorim conversia, în ordinea inversă obţinerii acestora.

```
TO CONVERSIE :N :B
IF :B < 2 [PRINT [BAZA DE CONVERSIE TREBUIE >=2] STOP]
IF :B > 16 [PRINT [BAZA DE CONVERSIE TREBUIE < 16] STOP]
IF :N = 0 [OUTPUT 0]
OUTPUT CONV :N :B
END

TO CONV :N :B
IF :N = 0 [OUTPUT "= ]
OUTPUT WORD (CONV (INT DIV :N :B) :B) (CIFRA (REMAINDER :N :B))
END

TO CIFRA :R
IF :R < 10  [OUTPUT :R]
IF :R = 10  [OUTPUT "A]
IF :R = 11  [OUTPUT "B]
IF :R = 12  [OUTPUT "C]
IF :R = 13  [OUTPUT "D]
IF :R = 14  [OUTPUT "E]
IF :R = 15  [OUTPUT "F]
END
```

Procedura CIFRA are ca intrare restul curent obţinut în procesul de conversie. Acesta este pus în corespondenţă cu cifra de reprezentare în baza B.

Procedura CONV este cea care realizează efectiv conversia din baza 10 în baza B a numărului N.

Printr-o chemare recursivă, avînd de fiecare dată ca intrare cîtul dintre număr şi baza în care vrem să convertim, se generează resturile în ordinea inversă obţinerii lor. De fiecare dată se apelează procedura CIFRA de asociere a restului cu cifra de reprezentare în baza B.

Programul principal verifică dacă baza de conversie este <16. În caz contrar, se specifică un mesaj şi se abandonează procesul de conversie.

În caz afirmativ se verifică dacă numărul este 0.

Dacă numărul este diferit de 0 şi baza mai mică decît 16 se apelează procedura CONV ce realizează conversia propriu-zisă.

Exemplu:

```
PRINT CONV 29 2
=11101
```

12.3.5. Conversia dintr-o bază oarecare în altă bază

Programul de conversie poate fi generalizat, pentru a realiza conversia unui număr reprezentat într-o bază oarecare în altă bază.

Pentru a evita efectuarea calculelor în altă bază decît baza 10, procesul de conversie va consta în două etape şi anume:

etapa 1 — converteşte numărul din baza în care este reprezentat, în baza 10, procedura BAZA.ZEC

etapa 2 — converteşte reprezentarea în baza 10, obţinută în urma etapei 1, în baza în care se doreşte conversia finală, procedura ZEC. BAZA

Este necesar să existe o procedură care pune în corespondenţă valoarea în zecimal a unei cifre într-o bază oarecare cu codul ASCII al cifrei respective, pentru a putea realiza tipărirea ei (ZEC.ASCII).

De asemenea, trebuie să existe o procedură care avînd ca intrare caracterul ASCII al unei cifre într-o bază oarecare să genereze valoarea în zecimal corespunzătoare (ASCII.ZEC).

OBSERVAŢIE: Codul ASCII reprezintă codificarea binară a caracterelor, cifrelor şi semnelor speciale

Simbolurile de reprezentare a cifrelor în diverse baze le putem considera astfel:

baza k	baza 10	baza m
0	0	0
1	1	1
2	2	2
3	3	3
4	4	4
5	5	5
6	6	6
7	7	7
8	8	8
9	9	9
A	10	A
B	11	B
C	12	C
D	13	D
E	14	E
F	15	F
G	16	G
.	17	H
.	18	I
.	.	.

ASCII.ZEC ⟶ . ⟶ ZEC.ASCII

ZEC.ASCII ⟵ . ⟵ ASCII.ZEC



```
TO ZEC.ASCII :N
IF :N < 10 [OUTPUT :N]
OUTPUT CHAR 55+:N
END
```

Procedura ZEC.ASCII are ca intrare un număr zecimal N şi generează codul ASCII corespunzător unei baze de reprezentare oarecare.

Dacă N>10 trebuie generat unul dintre simbolurile A, B, C, ...

Avînd în vedere că A are codul 65, B are codul 66, etc rezultă că pentru valorile mai mari decît 10 trebuie să adunăm 55 pentru a ajunge la codul ASCII corespunzător literelor mari A, B, C, ...

```
TO ASCII.ZEC :A
IF NUMBERP :A [OUTPUT :A]
OUTPUT (ASCII :A) — 55
END
```

Procedura ASCII.ZEC are ca intrare caracterul ASCII al unei cifre într-o bază oarecare şi generează la ieşire valoarea zecimală corespunzătoare. Realizează funcţia inversă lui ZEC.ASCII.

```
TO ZEC.BAZA :N :BAZA
IF :N < :BAZA [OUTPUT ZEC.ASCII :N]
OUTPUT LIST ZEC.BAZA INT DIV :N :BAZA :BAZA
            ZEC.ASCII REMAINDER :N :BAZA
END
```

Procedura ZEC.BAZA converteşte un număr din baza 10 într-o bază oarecare BAZA.

ZEC.ASCII REMAINDER :N :BAZA va genera caracterul ASCII asociat restului împărţirii numărului N la baza de conversie BAZA. Deci va genera cifra curentă a numărului în baza BAZA.

ZEC.BAZA INT DIV :N :BAZA :BAZA realizează printr-un proces recursiv operaţiile de obţinere a cîturilor, pînă cînd cîtul curent este mai mic decît baza nouă de reprezentare.

```
TO      BAZA.ZEC :N :BAZA :P
IF      EMPTYP :N [OUTPUT 0]
OUTPUT  (:P * ASCII.ZEC LAST :N)+
        BAZA.ZEC BUTLAST :N :BAZA :P * :BAZA
END
```

Procedura BAZA.ZEC converteşte un număr N reprezentat în baza BAZA, în baza 10. Procedura se bazează pe următoarea relaţie:

$$\overline{(a_m a_{m-1} \ldots a_0)} = a_m \cdot b^{m-1} + a_{m-1} \cdot b^{m-1} + \ldots + a_1 \cdot b^1 + a_0$$

EXEMPLU:

$$176 = 1 \times 8 \times 8 + 7 \times 8 + 6 \times 1$$

Parametrul P, în momentul apelării rutinei va fi 1 iar în procesul recursiv de apelare va căpăta valorile BAZA, BAZA * BAZA, etc. ce constituie cantităţile cu care se înmulţeşte fiecare cifră de reprezentare în baza BAZAF.

```
TO CONVERSIE.BAZAI.BAZAF :N :BAZAI  :BAZAF
IF :BAZAF<2 [PRINT [BAZA≥2]STOP]
OUTPUT ZEC.BAZA BAZA.ZEC :N :BAZAI 1 :BAZAF
END
```

Procedura CONVERSIE.BAZAI.BAZAF constituie programul principal de conversie a unui număr din baza iniţială BAZAI în baza finală BAZAF.

Converteşte numărul N din baza iniţială BAZAI în zecimal apelînd procedura BAZA.ZEC după care se converteşte numărul zecimal obţinut, în baza finală BAZAF apelînd procedura ZEC.BAZA.

Ca exerciţiu, propunem cititorilor să rescrie procedura BAZA.ZEC bazîndu-se pe relaţia:

$$\overline{(a_m a_{m-1} \ldots a_0)} = ((..(a_m * b + a_{m-1}) * b + \ldots + a_1) * b + a_0)$$

Ca exemplu de utilizare a procedurii de conversie generalizată arătăm cum se foloseşte în realizarea conversiei din zecimal în hexa şi invers.

```
TO ZECIMAL.HEX :N
OUTPUT CONVERSIE.BAZAI.BAZAF :N 10 16
END

TO HEX. ZECIMAL  :N
OUTPUT CONVERSIE.BAZAI.BAZAF :N 16 10
END
```

12.3.6. Desenează interactiv

Să se scrie un program LOGO care are următoarele specificaţii:

— citeşte starea claviaturii şi în funcţie de tasta apăsată execută una din comenzile:

F — deplasează penelul înainte cu valoarea 10
R — roteşte dreapta penelul cu 30 grade
L — roteşte stînga penelul cu 30 grade
C — şterge ecranul
S — anulează ultima comandă executată
N — defineşte o figură
D — desenează o figură al cărei *nume* se specifică
À — afişează programul asociat unei figuri
M — afişează meniul de comenzi

— în funcţie de comanda specificată de la tastatură execută operaţia asociată şi în acelaşi timp generează o listă de primitive ce în final va constitui un program LOGO.

Utilizatorul nu va trebui să scrie un program care să deseneze o figură, ci acesta va fi generat automat în urma mişcării penelului pe ecran cu comenzile F, R, L, D, etc.

```
TO MENIU
   TEXTSCREEN
   CLEARTEXT
   PRINT [F:   DEPLASEAZĂ PENELUL ÎNAINTE 10 PUNCTE]
   PRINT [R:   ROTEŞTE PENELUL DREAPTA CU 30 GRADE]
   PRINT [L:   ROTEŞTE PENELUL STÎNGA CU 30 GRADE]
   PRINT [C:   ŞTERGE ECRANUL (CLEARSCREEN)]
   PRINT [S:   ŞTERGE ULTIMA COMANDĂ]
   PRINT [N:   DEFINEŞTE O FIGURĂ]
   PRINT [D:   DESENEAZĂ O FIGURĂ SPECIFICATĂ]
   PRINT [A:   AFIŞEAZĂ PROGRAMUL REZULTAT PENTRU FIGURA]
   PRINT [M:   AFIŞEAZĂ MENIUL]
   PRINT [ ]   PRINT [ ]
   PRINT [     APĂSAŢI ORICE TASTĂ PENTRU A CONTINUA]
   PRINT       READCHAR
   END
```

Procedura MENIU (asociată comenzii M) afişează pe ecran comenzile asociate diferitelor taste F, R, L, etc.

Pentru a scrie programul este necesar să scriem cîte o procedură pentru fiecare comandă în parte

```
TO ŞTERGE.ECRAN
   CLEARSCREEN
   MAKE "FIGURA [ ]
END
```

Procedura ŞTERGE.ECRAN (asociată comenzii C) are ca efect ştergerea ecranului în regim grafic şi anularea listei de primitive ce se asociază figurii ce se desenează.

Variabila FIGURA este o listă ce va păstra istoria comenzilor ce se execută în timpul desenării unei figuri.

În final se va constitui programul ce va desena figura, care a fost pregătită în mod interactiv.

```
TO ANULEAZĂ.COM
   IF :FIGURA = [ ] [STOP]
   MAKE "FIGURA BUTLAST :FIGURA
   CLEARSCREEN
   TRASEAZĂ.FIG :FIGURA
END
```

Procedura ANULEAZĂ.COM (asociată comenzii S) şterge ultima comandă care a fost introdusă (eliminînd primitiva din lista asociată figurii) şterge ecranul şi redesenează figura.

```
TO TRASEAZĂ.FIG :COMENZI
IF :COMENZI = [ ] [STOP]
RUN FIRST :COMENZI
TRASEAZĂ.FIG BUTFIRST :COMENZI
END
```

Procedura TRASEAZĂ.FIG trasează o figură, executînd comandă cu comandă dintr-o listă de comenzi specificată.

```
TO NUME.FIG
   PRINT [CARE ESTE NUMELE FIGURII?]
TRASEAZĂ.ŞI.ÎNREG FPUT FIRST READLIST [ ]
END
```

Procedura NUME.FIG (asociată execuţiei comenzii D — desenează o figură al cărei nume a fost specificat) cere utilizatorului numele figurii ce se doreşte să fie afişată şi o trasează.

```
TO TRASEAZĂ.ŞI.ÎNREG :ACŢIUNE
   RUN :ACŢIUNE
   MAKE "FIGURA (LPUT :ACŢIUNE :FIGURA)
END
```

Procedura TRASEAZĂ.ŞI.ÎNREG (asociată comenzilor F, R, L) are ca efect executarea acţiunii asociată comenzilor F, R, L, şi înregistrarea primitivei corespunzătoare în lista ce va constitui în final programul de desenare a figurii.

```
TO ÎNVAŢĂ
   PRINT [CUM SE NUMEŞTE PROGRAMUL ASOCIAT FIGURII?]
   MAKE "NUME (FIRST READLIST)
   DEFINE :NUME (FPUT [ ] :FIGURA)
   ŞTERGE.ECRAN
END
```

Procedura ÎNVAŢĂ (asociată comenzii N) cere utilizatorului numele programului ce se va crea pe baza listei de comenzi creată interactiv. Se creează programul ce generează figura respectivă după care se şterge ecranul, şi se aşteaptă începutul unei noi sesiuni de lucru pentru definirea unei alte figuri.

```
TO AFIS.PROG
TEXTSCREEN
PRINT [NUMELE PROGRAMULUI ASOCIAT FIGURII]
MAKE "NUME FIRST READLIST
PO :NUME
END
```

Procedura AFIS.PROG (asociată comenzii A) afişează programul LOGO rezultat pentru figura desenată interactiv şi salvată cu comanda N.

```
TO EXECUTĂ.COMANDA :COM
IF :COM = "F [TRASEAZĂ.ŞI.ÎNREG [FORWARD 10] STOP]
IF :COM = "R [TRASEAZĂ.ŞI.ÎNREG [RIGHT 30] STOP]
IF :COM = "L [TRASEAZĂ.ŞI.ÎNREG [LEFT 30] STOP]
IF :COM = "C [ŞTERGE.ECRAN STOP]
IF :COM = "S [ANULEAZĂ.COM STOP]
IF :COM = "N [ÎNVATĂ STOP]
IF :COM = "D [NUME.FIG STOP]
IF :COM = "A [AFIS.PROG STOP]
IF :COM = "M [MENIU STOP]
END
```

Procedura EXECUTA.COMANDA are ca intrare numele comenzii ce trebuie executată (F, R, L, C, S, N, D, A, M), asociază şi execută procedura corespunzătoare.

```
TO EXECUTĂ.COMENZI
   EXECUTA.COMANDA READCHAR
   EXECUTĂ.COMENZI
END
```

Procedura EXECUTA.COMENZI execută comanda ce a fost specificată de utilizator de la tastatură şi repetă acest proces.

```
TO DESENEAZĂ
   MENIU
   ŞTERGE.ECRAN
   EXECUTĂ.COMENZI
END
```

Procedura DESENEAZĂ constituie programul principal care implementează specificaţiile precizate.

La început afişează meniul de comenzi utilizabile în program, iniţializează lista de primitive pe baza căreia se va genera programul de desenare a figurii, şterge ecranul şi cheamă rutina ce va executa comenzile specificate de utilizator.

12.3.7. Turnurile din Hanoi

Se consideră 3 tije denumite STÎNGA, MIJLOC, DREAPTA şi n discuri de dimensiuni diferite, aşezate pe tija din STÎNGA, în ordine descrescătoare a dimensiunilor lor, formînd un "turn" ca în fig. 12.6.

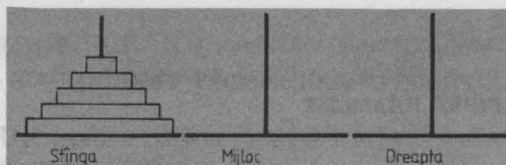


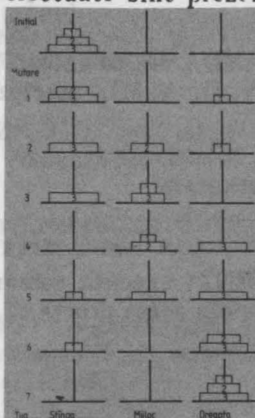
Fig. 12.6 Poziţia iniţială a discurilor

Se pune problema să mutăm cele n discuri din poziţia iniţială STÎNGA în poziţia finală DREAPTA, respectînd regulile:
— la fiecare mişcare se mută un singur disc;
— un disc nu poate fi aşezat peste unul mai mic decît el;
— tija MIJLOC se poate folosi pentru mutări intermediare.

Rezolvare:

Pentru început să exemplificăm mutările pentru cazul n=3, în vederea stabilirii unui algoritm. Presupunem discurile notate cu 1, 2, 3 şi vom considera că dimensiunile sînt puse în corespondenţă cu aceste numere, adică discul 1 este cel mai mic iar discul 3 este cel mai mare.

Mutările ce trebuie efectuate sînt prezentate în fig. 12.7.


Fig. 12.7. Mutările efectuate pentru cazul n=3 discuri.

Se observă că pentru a putea muta discul cel mai mare (în cazul nostru 3) de pe tija iniţială STÎNGA, pe tija finală DREAPTA este necesar să luăm toate discurile de deasupra lui (2 şi 1) să şi le mutăm pe tija MIJLOC, folosindu-ne de tija DREAPTA ca tijă intermediară. În figura 12.7 se ajunge în această situaţie prin efectuarea primelor trei mutări.

Problema mutării celor n discuri din poziţia iniţială STÎNGA în poziţia finală DREAPTA poate fi redusă la problema mutării a (n-1) discuri din poziţia STÎNGA în poziţia MIJLOC, folosind ca tijă intermediară DREAPTA.

Aceasta conduce la ideea realizării unei proceduri [mutare] ce va muta cele n discuri, care se va apela pe ea însăşi ca să mute n-1 discuri, ş.a.m.d. Se observă că problema se poate rezolva prin recursivitate.

Mutarea efectivă a unui disc pe tija finală DREAPTA se poate efectua numai cînd numărul de discuri pe tija sursă (cu care lucrează procedura la un moment dat, în timpul apelării recursive) este 1.

Procedura de mutare efectivă a unui disc este:

```
TO    mutadisc :s :d
      (TYPE [Muta un disc din] "\ :s "\ in \ :d)
      PRINT"
END
```

Pentru a indica efectuarea mutării se va afişa un mesaj:

Muta un disc din SURSA în DESTINAŢIE.

Operaţiile ce trebuie efectuate de procedura ce realizează mutările [mutare] sînt următoarele:

— mută primele n-1 discuri din poziţia STÎNGA în poziţia MIJLOC, folosind tija DREAPTA ca tijă intermediară, prin apelare recursivă [continuă mutare];

— mută discul rămas pe tija STÎNGA în poziţia finală DREAPTA, cînd apelarea procedurii se face pentru un disc, [muta disc];

În acest fel, unul din discuri ajunge în poziţia finală şi va trebui să ne ocupăm de celelalte n-1, pentru a le transfera de pe tija MIJLOC pe tija DREAPTA, prin intermediul tijei STÎNGA (situaţie obţinută în urma mutării 4 în exemplul considerat).

Procesul continuă ca şi cum ne-am afla în faza iniţială numai că sînt mai puţine discuri (n-1) iar tijele sînt MIJLOC, STÎNGA, DREAPTA.

Procedura de efectuare mutare este:

```
TO    mutare :n :s :m :d
      IF :n=1 [mutadisc :s :d STOP] [continuă_mutare]
END
```

unde cei patru parametri au următoarea semnificaţie:

:n — numărul de discuri de pe tija curentă;
:s — tija STÎNGĂ;
:m — tija MIJLOC;
:d — tija DREAPTA;

Dacă numărul de discuri pe tija curentă este 1 atunci mută discul respectiv în poziţia finală, prin execuţia procedurii [mutadisc], iar în caz contrar continuă cu mutarea a n-1 discuri prin apelarea procedurii [continuă mutare].

Procedura [continuă mutare] realizează operaţiile:

— mută cele n-1 discuri de pe tija STÎNGA pe tija MIJLOC folosind ca tijă intermediară tija DREAPTA;

— mută efectiv discul din STÎNGA în DREAPTA;

— mută cele n-1 discuri, din cele rămase, de pe tija MIJLOC pe tija DREAPTA folosind, ca tijă intermediară, tija STÎNGA.

Procedura continuă mutare este:

```
TO    continua_mutare
      mutare :n-1 :s :d :m
      mutad'sc :s :d
      mutare :n-1 :m :d
END
```

Numărul de mutări necesar pentru transferul celor n discuri de pe tija STÎNGA pe tija DREAPTA folosind, ca tijă intermediară tija MIJLOC, este 2^n-1.

Programul final care interacţionează cu utilizatorul în vederea stabilirii numărului de discuri şi contorizează mutările efectuate este:

```
TO MUTADISC :s :d
MAKE "nr_mutare :nr_mutare +1
(TYPE :nr_mutare" [Muta din] "\ :s " \ in \ :d)
PRINT"
END

TO CONTINUA_MUTARE
MUTARE :n-1 :s :d :m
MUTADISC :s :d
MUTARE :n-1 :m :s :d
END

TO MUTARE :n :s :m :d
IF :n=1 (mutadisc :s :d stop) (continua_mutare)
END

TO HANOI
TYPE (Introduceţi număr discuri :)
MAKE " nr_discuri READCHAR
TYPE :nr_discuri
MAKE "nr_mutare 0
PRINT"
MUTARE :nr_discuri "stinga "mijloc "dreapta
END
```

Exemplu de utilizare:

2 hanoi

Introduceţi număr discuri **:3**

1 *Mută din stinga în dreapta*
2 *Mută din stinga în mijloc*
3 *Mută din dreapta în mijloc*
4 *Mută din stinga în dreapta*
5 *Mută din mijloc în stinga*
6 *Mută din mijloc în dreapta*
7 *Mută din stinga în dreapta*

2.4. Mesaje de eroare generate de LOGO

Varianta în limba engleză	Varianta în limba română
Not enough input to ...	Mai trebuie intrări ...
I don't know how to ...	Nu ştiu cum să fac ...
You don't say what to do with ...	Nu ai spus ce să fac cu ...
... does not output to ...	... nu are ieşiri în ...
... is used by Logo	... utilizat de Logo
... is already defined	... este deja definit
... is not true or false	... nu este adevărat / fals
... is not word	... nu este cuvînt
Too many inside parentheses	Prea multe paranteze
... open file problem	... probleme la open
... file not found	... fişier negasit
Bad file name	Nume greşit
You're at toplevel	Sînteţi în comandă
STOPPED	ANULARE
Turtle out of field	Penel afară ecran
Not enough space to proceed	Nu e spaţiu să prelucrez
... doesn't like	... nu îi place
... has no value	... nu are val.
... is a primitive	... e o primitivă
Not enough items in ...	Mai puţine elem. în ...
Overflow	Depăşire
... can't divide by zero	... nu pot împărţi cu 0
... number too big	... număr prea mare
... as imput	... ca intrări
... in	... în

Obs. Traducerea în limba română este făcută în aşa fel încît să se încadreze în acelaşi număr de caractere.

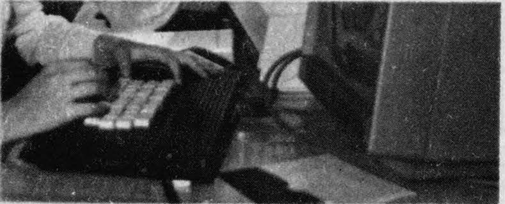
• Cartea se poate subintitula „Totul despre... HC-85", HC-85 fiind primul calculator personal românesc compatibil cu o familie larg răspîndită pe plan internațional (Sinclair-Spectrum). Ea continuă ciclul început în 1985 cu „Totul despre... calculatorul personal aMIC" și continuat în 1989 cu „Totul despre... microprocesorul Z80" și „Totul despre... BASIC", în redacția de informatică și tehnică de calcul a Editurii Tehnice.

• Este scrisă de specialiști din cercetare-proiectare-fabricație-învățămînt, de la catedra de calculatoare a Institutului Politehnic București — Facultatea de Automatică (conceptori-proiectanți ai lui HC-85, în frunte cu Prof. dr. ing. Adrian Petrescu, coordonatorul colectivului), respectiv de la Fabrica de Calculatoare Electronice (care a realizat calculatorul HC-85), de la Institutul de Tehnică de Calcul (ITC), Institutul de Cercetări pentru Informatică (ICI), Liceul Dimitrie Cantemir, ș. a. (reprezentanți din cei mai autorizați ai realizatorilor de software, inclusiv de programe de aplicații pentru HC-85, ai învățămîntului elementar și mediu și centrelor de instruire pentru diverse categorii de utilizatori de calculatoare, cum și ai editorilor de specialitate).









• Este alcătuită din două volume (XII părți, 25 capitole), o parte a tirajului fiind însoțită de 3 casete magnetice încărcate cu programe (interpretor LOGO și exemple LOGO, interpretor BETA BASIC și lecții BASIC, lecții de structură și utilizare a lui HC-85, programe de aplicații în BASIC etc.), ce acționează / utilizează calculatoare HC-85 sau compatibile.

• După un studiu introductiv al Acad. prof. Nicolae Teodorescu și o secțiune intitulată PROLOG-DIALOG-EPILOG (ce continuă la începutul și sfîrșitul volumului 2) primul volum cuprinde: I. Calculatoare, microcalculatoare și calculatoare personale în țara noastră și pe plan mondial. II. Calculatoare numerice. Realizare fizică — baze aritmetice și logice, III. Calculatorul personal HC-85. Structură, componente, operare, programare. IV. Programarea în BASIC pe HC-85. V. Programarea în LOGO pe HC-85, volumul asigurînd cunoștințe fundamentale și aplicative profunde, ultima parte fiind și o premieră în literatura românească.

ISBN 973-31-0013-7 Volumul I și II Lei 89 ISBN 973-31-0014-5